BACHELORARBEIT

Handling einer Plattform-Restbussimulation im Automotive-Bereich mit einer dynamischen Anpassung an spezifische Kunden

Lorenz Brüstle

2020

Lorenz Brüstle

Handling einer Plattform-Restbussimulation im Automotive-Bereich mit einer dynamischen Anpassung an spezifische Kunden

Bachelorarbeit für die Prüfung zum Bachelor of Engineering (B.Eng.)

Im Studiengang Elektrotechnik/Informationstechnik
Fakultät Elektrotechnik, Medizintechnik und Informatik
an der Hochschule für Technik, Wirtschaft und Medien Offenburg

Betreuender Prüfer: Dipl.-Math., Dipl.-Biol. Joachim Jauß
Zweitgutachter: Prof. Dr.-Ing. Daniel Fischer

Bearbeitungszeitraum: 01.09.2019 – 29.02.2020

Kurzfassung

Da die hohe Anzahl an Steuergeräten in einem Fahrzeug von den unterschiedlichsten Automobilzulieferern entwickelt und produziert werden, ist es den einzelnen Steuergeräte-Herstellern nicht möglich, diese während des Entwicklungsprozesses in einem realen Fahrzeug zu testen. Restbussimulationen, womit Fahrzeugnetzwerke softwaretechnisch nachgebaut werden, schaffen hierbei Abhilfe.

Für die Entwicklung konkurrenzfähiger, effizienter und wirtschaftlicher Steuergerätesoftware wird die Embedded Software in einzelne Module unterteilt. Dieser modulare Prozess ermöglicht das Implementieren der Embedded Softwaremodule in Steuergeräte unterschiedlicher Fahrzeughersteller, sodass es zu Kosteneinsparungen während der Entwicklung und Wartung kommt. Steuergeräte, welche in unterschiedlichen Fahrzeugen zum Einsatz kommen und eine hohe Anzahl an gleichen Softwaremodulen besitzen, werden in sogenannten „Plattformen“ gehandelt.

Im Rahmen dieser Arbeit wird, analog zu der Plattformsoftware der Steuergeräte eine Plattform-Restbussimulation entworfen. Sie stellt dem Softwareentwickler während des kompletten Entwicklungszyklus eine lauffähige Testumgebung zur Verfügung, welche wichtige Steuergeräte eines Fahrzeugnetzwerks nachbildet. So werden in dieser Arbeit Konzepte erstellt und implementiert, welche eine effiziente und intuitive Benutzung der Plattform-Restbussimulation ermöglichen und alle Plattformkunden mit einer einzigen Simulationsumgebung abdecken. Dies führt zu einer zeitlichen Einsparung bei der Implementierung, Verwaltung und Bedienung.

Abstract

As the high numbers of ECUs in a vehicle are developed and produced by many different automotive suppliers, it is not possible for each manufacturer to test the ECU during the development cycle in a real vehicle. The solution is a residual bus simulation to imitate a vehicle network.

For the development of competitive, efficient and economic ECU software, the embedded software is divided into separate modules. This modular process creates the possibility to implement the embedded software modules into ECUs of different automotive manufacturers. This enables that the suppliers of ECUs reduce the costs during the development and maintenance. ECUs, which are used in different vehicles and have a high number of equal embedded software modules, are handled in so-called “platforms”.

In this work a platform for residual bus simulation was developed. It was constructed analogously to the software platform of the ECUs. The residual bus simulation provides the developers a runnable test environment during the complete development cycle which simulates important ECUs of the vehicle network. In this work concepts are created and implemented which enable an efficient and intuitive use of the simulation environment and cover all customers of the platform with one simulation. This reduces the time spent on the implementation, administration and usage of the bus simulation platform.

Inhaltsverzeichnis

Nomenklatur **VI**

Abbildungsverzeichnis **VIII**

Tabellenverzeichnis **X**

Listings **XI**

1 Einleitung **1**

1.1 Problemstellung ... 2

1.2 Zielsetzung ... 2

1.3 Vorgehen ... 3

2 Grundlagen **4**

2.1 Einführung in CANoe ... 4

2.2 Restbussimulation ... 9

2.3 Bussysteme im Automotive-Bereich ... 10

2.3.1 Controller Area Network – CAN ... 12

2.3.2 SAE J1939 Protokoll ... 16

2.3.3 Diagnoseprotokoll ... 18

2.4 Keyless ... 21

3 Anforderungen an die Plattform-RBS **24**

3.1 Funktionale Anforderungen ... 24

3.2 Nichtfunktionale Anforderungen ... 26

4 Konzeptentwicklung für die Plattform-Restbussimulation **28**

4.1 Aufbau der Plattform-Restbussimulation ... 28

4.2 Dynamische Protokollanpassung ... 33

4.3 Dynamisches Handling der Hersteller ... 36

4.4 Einbinden der Diagnosedienste ... 40

5 Implementierung 43
5.1 Aufteilung in separate Herstellernetzwerke 43
5.1.1 Treibereinstellungen 44
5.1.2 Qualifizierte Signale und Botschaften 45
5.1.3 Deaktivieren von CAPL-Knoten/Netzwerken 48
5.2 Dynamische Anpassung der Protokolle CAN und J1939 48
5.3 Grafische Auswahl der Hersteller 49

6 Bewertung und Inbetriebnahme der Restbussimulation 55
6.1 Bewertung der funktionalen Anforderungen 56
6.2 Bewertung der nichtfunktionalen Anforderungen 58

7 Fazit und Ausblick 60

Literaturverzeichnis 62

Nomenklatur

Abkürzungsverzeichnis

ABS	Antiblockiersystem
Bus	Binary Unit System
CAN	Controller Area Network
CAPL	CAN Access Programming Language
DTC	Diagnostic Trouble Code
ECU	Electronic Control Unit (Steuergerät im Fahrzeugbereich)
HF	High Frequency
ISO	International Organization for Standardization
LF	Low Frequency
ms	Millisekunden
OBD	On-Board-Diagnose
OEM	Orginal Equipment Manufacturer
OSI	Open System Interconnection
RBS	Restbussimulation

Begriffe und Definitionen

Botschaft	Beschreibt innerhalb des CAN-Busses die Struktur und den Aufbau der CAN-Nachricht.
Bus	Topologie, an dem alle Netzwerkknoten an einem gemeinsamen Übertragungsmedium angeschlossen sind.
CAN-Frame	Laut ISO 11898-1 beinhaltet der CAN-Frame alle Informationen, welche für die Abwicklung des CAN-Kommunikationsprotokolls erforderlich sind, um eine Datenübertragung zu gewährleisten.
OEM	Im Automotive-Bereich werden Fahrzeughersteller synonym dazu verwendet. Zudem werden in dieser Arbeit die Begriffe Kunde und Hersteller analog zum OEM betrachtet.
Signal	Gibt die Nutzdaten eines CAN-Frames an. Dabei können Signale einzelne Bits sowie komplette Bytes beschreiben.

Abbildungsverzeichnis

Abbildung 2.1: Visuelle Darstellung der Zuordnung von Signalen zu einer Botschaft innerhalb der Datenbank..... 6
Abbildung 2.2: Entwicklungsprozess einer Restbussimulation mit CANoe [CANoe 2018, S. 20]..... 9
Abbildung 2.3: Bussysteme im modernen Fahrzeug [Zimmermann und Schmittgall 2014, S. 2]..... 11
Abbildung 2.4: Standard Daten Frame..... 13
Abbildung 2.5: Beschreibung eines CAN Error-Frames [CANBUS 2018]..... 16
Abbildung 2.6: Protokollstapel nach SAE J1939 [Zimmermann und Schmittgall 2014, S. 178]..... 17
Abbildung 2.7: J1939-Identifier..... 18
Abbildung 2.8: Kommunikationsmodell der KWP 2000-Diagnose [Zimmermann und Schmittgall 2014, S. 192]..... 20
Abbildung 2.9: "Seed & Key"-Verfahren [Borgeest 2014, S. 276]..... 21
Abbildung 2.10: Funktionaler Bereich des Keyless-Schlüssels für die Ent- und Verriegelung eines Fahrzeugs [HELLA 2019]. 22
Abbildung 3.1: Protokoll-Definition (J1939-Protokoll) in der Datenbank für das Netzwerk J1939_OEM_3 25
Abbildung 4.1: Alle Netzwerkknoten der Plattform werden in einem Netzwerk verwaltet 29
Abbildung 4.2: Inaktives Netzwerk für den ersten Plattform OEM..... 31
Abbildung 4.3: Aktives Netzwerk für den zweiten Plattform OEM 31
Abbildung 4.4: Globales Netzwerk, welches die Netzwerkknoten der Plattform enthält..... 32
Abbildung 4.5: Netzwerkattribute für die Definition des Bus- und Protokolltyps 34
Abbildung 4.6: Zuweisung der kundenspezifischen Diagnosebeschreibung für die einzelnen OEM-Netzwerke 42

Abbildung 5.1: Die hinzugefügten Kanäle (CAN 1 - globales Netzwerk und CAN 4 - kundenspezifisches Netzwerk) entsprechen je einem Netzwerk innerhalb CANoe. 45
Abbildung 5.2: Kanalzuweisung für redundante Botschaften über Namensraumdefinition 46
Abbildung 5.3: Grafisches Panel für die Auswahl der spezifischen Kundenparameter sowie das angeschlossene ECU. 50

Tabellenverzeichnis

Tabelle 2.1: Vor- und Nachteile von CAPL gegenüber der Hochsprache C [Lobmeyer und Marktl 2014 b] 7
Tabelle 2.2: ISO/OSI-Referenzmodell des CAN-Bus 12
Tabelle 4.1: Vor- und Nachteile für den Ansatz 4.1.1 – Anlegen eines globalen Netzwerks 30
Tabelle 4.2: Vor- und Nachteile für den Ansatz 4.1.2 – Aufteilung in separate Netzwerke 33
Tabelle 4.3: Vor- und Nachteile für den Ansatz 4.2.1 – Manuelle Anpassung der Übertragungsprotokolle 34
Tabelle 4.4: Vor- und Nachteile für den Ansatz 4.2.2 – Dynamische Anpassung der Übertragungsprotokolle 35
Tabelle 4.5: Vor- und Nachteile für den Ansatz 4.3.1 – Manuelle Auswahl des Kunden über grafische Benutzeroberfläche 37
Tabelle 4.6: Vor- und Nachteile für den Ansatz 4.3.2 – Automatische Auswahl des Kunden über die ECU-Softwarenummer 38
Tabelle 4.7: Vor- und Nachteile für den Ansatz 4.3.3 – Automatische Auswahl des Kunden über eine Diagnoseanfrage 40
Tabelle 4.8: Vor- und Nachteile für den Ansatz 4.4.1 – Setzen der Diagnosebeschreibung beim Simulationsstart 41
Tabelle 4.9: Vor- und Nachteile für den Ansatz 4.4.2 – Jedem Herstellernetzwerk wird eigene Diagnosebeschreibung zugewiesen 42
Tabelle 6.1: Bewertung der funktionalen Anforderungen der Restbussimulation für zwei Kunden. 58
Tabelle 6.2: Bewertung der nichtfunktionalen Anforderungen der RBS 59

Listings

Listing 4.1: Auslesen der Softwarenummer für den Kunden Mahindra..... 37
Listing 4.2: Response-Funktion der ECU-Softwarenummer für Mahindra..... 38
Listing 4.3: Anfrage für eine Diagnosesitzung 39
Listing 4.4: Response-Funktion für die Diagnosesitzung 39
Listing 5.1: Namensraumdefinition über sogenannte „Qualifier“ 46
Listing 5.2: Verweis auf entsprechende Datenbank 46
Listing 5.3: Verweis auf entsprechenden softwareseitigen Netzwerkkanal 47
Listing 5.4: Verweis auf das kundenspezifische Netzwerk 47
Listing 5.5: Einlesen aller Datenbanken, welche sich in einem Ordner befinden..... 49
Listing 5.6: Abfrage der Umgebungsvariable in Bezug auf den eingestellten Kunden 51
Listing 5.7: Manuelles Senden von Diagnoseanfragen 52

1 Einleitung

Bereits bei den ersten mikroprozessorgesteuerten Systemen 1980 wie Motronic und ABS in Kraftfahrzeugen fand ein Datenaustausch zwischen diesen Systemen statt. Durch die große Anzahl an herstellerspezifischen Lösungen für den Datenaustausch und vor allem für Diagnosetests für Steuergeräte in den Kfz-Werkstätten wurde schnell nach einer herstellerübergreifenden Lösung gesucht. So wurde bereits 1990 der CAN-Bus unter der ISO 11898 von BOSCH eingeführt, welcher selbst in heutigen hochkomplexen Fahrzeugen für den Großteil des Datenflusses zwischen Steuergeräten verantwortlich ist. Im Laufe der Zeit wurden die Systeme immer komplexer und das Datenaufkommen größer, sodass weitere Bussysteme im Kraftfahrzeug Einsatz fanden und nach unterschiedlichen Standards implementiert wurden [Zimmermann und Schmittgall 2014, S. 1]. Diese standardisierte Kommunikation zwischen den Steuergeräten ermöglicht den Steuergeräte-Herstellern eine modulbasierte Entwicklung, welche in den unterschiedlichen Fahrzeugen eingesetzt werden kann, da die Protokolle und Schnittstellen spezifiziert wurden. Durch diese modulbasierte Entwicklung wurde eine deutliche Kostenreduktion erreicht, da die Entwicklungszeit und der Verwaltungsaufwand reduziert werden konnten.

Eine weitere, nicht zu vernachlässigende Komponente während des Entwicklungszyklus der Steuergeräte sind die Testphasen. Ob bewährte V-Modelle [Ernst u.a. 2015, S. 756] oder agile Entwicklungsprozesse wie SCRUM [Sutherland und Schwaber 2017], jedes Modell verlangt ausgiebige Tests, um frühzeitig Fehler erkennen und beheben zu können.

1.1 Problemstellung

Damit Hersteller von Kfz-Steuergeräten diese während der Entwicklungsphase testen können (Softwaretests), kommen Simulationsumgebungen zum Einsatz, mit denen sich die relevanten Steuergeräte, welche für den unmittelbaren Datenaustausch nötig sind, nachbilden lassen. Zudem soll hierdurch die volle Funktionsfähigkeit des entwickelten ECU getestet werden können. Da es sich bei der verwendeten Topologie für die Datenübertragung um ein Bussystem handelt, spricht man hier von einer Restbussimulation, welche dem Entwickler eine funktionsfähige Umgebung für ein zu entwickelndes Steuergerät bereitstellt. Diese Restbussimulationen ermöglichen dem Entwickler neu implementierte Software ausgiebig zu testen, ohne dabei auf ein reales Kraftfahrzeug angewiesen zu sein. ECU-Kunden, welche eine hohe Redundanz an Softwaremodulen aufweisen, werden in einer „Plattform“ verwaltet. Um ebenfalls Einsparungen bei der Simulationsumgebung zu erlangen, bietet es sich an, eine Plattform-Restbussimulation zu verwenden, welche alle Kunden innerhalb der Plattform abdeckt. Jedoch wird in der Praxis jeder Kunde mit einer spezifischen Restbussimulation getestet. Dies stellt einen erhöhten Verwaltungsaufwand dar, da selbst gleiche Module innerhalb der verschiedenen Simulationsumgebungen mehrfach implementiert und gepflegt werden müssen, was der Wirtschaftlichkeit eines Unternehmens entgegenwirkt.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, eine Plattform-Restbussimulationsumgebung zu planen, zu entwickeln und zu implementieren. Diese Simulation ist dafür ausgelegt, verschiedene OEMs, welche die gleichen Softwaremodule verwenden, während des kompletten Entwicklungsprozesses eines Steuergeräts zu unterstützen. Das gemeinsame Handling der unterschiedlichen OEMs [OEM 1985] soll den Verwaltungsaufwand der Restbussimulation während der Entwicklungsphase verringern. Zudem soll das neue Konzept eine

benutzerfreundliche Bedienung sicherstellen und das Einbinden von weiteren, zukünftigen Plattformkunden ermöglichen. Eine automatische und dynamische Anpassung an die zu testenden kundenspezifischen ECUs soll eine effiziente Benutzung ermöglichen und manuelle Eingriffe, welche Fehlerquellen darstellen, weitestgehend ausschließen.

1.3 Vorgehen

Kapitel 2 beinhaltet die Grundlagen, die für das Verständnis dieser Arbeit unabdingbar sind. Dabei wird auf die Simulationsumgebung und verschiedene Busprotokolle, welche in Kraftfahrzeugen vorkommen sowie das Keyless-Prinzip eingegangen.

Kapitel 3 enthält die spezifischen Anforderungen, welche an die Plattform-Restbussimulation gestellt werden.

Die verschiedenen Konzepte, um die Simulationsumgebung optimal für den Benutzer unter Ausschöpfung der vorhandenen Möglichkeiten aufzubauen, werden in Kapitel 4 beschrieben. Dabei wird individuell auf die Vor- und Nachteile der unterschiedlichen Konzepte eingegangen.

Die Umsetzung der geeigneten Konzepte wird in Kapitel 5 aufgeführt.

Um die Funktionsfähigkeit der Restbussimulation sicherzustellen, wird diese für die spezifischen Plattformhersteller einer Testreihe unterzogen und in Kapitel 6 dargestellt. In der ersten Testphase werden grundlegende Tests durchgeführt, um wichtige sowie sicherheitskritische Funktionen und Module zu testen. Im zweiten Schritt findet das Ausrollen der Simulationsumgebung statt. Dabei testen die einzelnen Modulentwickler ihre spezifischen Anwendungen.

In Kapitel 7 wird ein Fazit der Arbeit gezogen. Ein anschließender Ausblick gibt Auskunft darüber, welche Erweiterungen für die Plattform-RBS künftig getätigt werden können.

2 Grundlagen

Dieses Kapitel beinhaltet die notwendigen Grundlagen, welche zum Verständnis dieser Arbeit benötigt werden. Kapitel 2.1 beinhaltet die Einführung in das Softwaretool CANoe, mit der die Plattform-Restbussimulation aufgebaut wird. Nachfolgend in Kapitel 2.2 wird auf die Funktionsweise und den Aufbau einer Restbussimulation eingegangen. Über welche Bussysteme die Steuergeräte im Automotive-Bereich kommunizieren und welche Übertragungsprotokolle in dieser Arbeit zum Einsatz kommen, wird in Kapitel 2.3 beschrieben. Abschließend wird auf die Keyless-Funktion näher eingegangen, für die das entwickelte Steuergerät zuständig ist.

2.1 Einführung in CANoe

Das Softwaretool der Firma Vector „CANoe“ wird für die Entwicklung, Tests und Analyse von Steuergeräten eingesetzt. Mit dieser Software können komplette Netzwerke, zum Beispiel ein CAN-Netzwerk, wie es in einem Kraftfahrzeug vorkommt, nachgebildet werden. So wird die Restbussimulation mit diesem Softwaretool umgesetzt. Dabei unterstützt die Software Netzwerkdesigner, Entwicklungs- und Testingenieure im kompletten Entwicklungsprozess – von der Planung bis hin zur Inbetriebnahme kompletter, verteilter Systeme oder einzelner Steuergeräte [CANoe 2018, S. 5]. Für eine komfortable, leichte sowie übersichtliche Handhabung der Simulation bietet CANoe externe Datenbasen, grafische Benutzeroberflächen, eine eigene ereignisgesteuerte Programmiersprache sowie weitere grafische und interaktive Fenster, welche die Auswertung der Bussysteme erleichtern. Zudem werden neben CAN noch

weitere Übertragungsprotokolle, wie z. B. CAN-FD, LIN, FlexRay und Ethernet durch CANoe unterstützt [CANoe 2018, S. 6].

Globale Datenbanken

Die speziell für CANoe entworfene Datenbank beschreibt das jeweilige Netzwerk. Die Netzwerkbeschreibung enthält wichtige Informationen wie Bustyp, Protokolltyp, Netzwerkname, Intel- oder Motorola-Format[1] sowie Baudrate. Zudem ermöglichen die Datenbanken das Beschreiben von Steuergeräten, Botschaften, Signalen und Umgebungsvariablen. Hierdurch ist eine symbolische Darstellung der definierten Informationen in CANoe möglich [CANoe 2018, S. 10, Lobmeyer und Marktl 2014 c].

- **Umgebungsvariablen** dienen als globaler Zugriffspunkt in CANoe. So ist ein Zugriff über grafische Oberflächen oder Netzwerkknoten auf die globalen Inhalte möglich. Wertetabellen erlauben die textuelle Beschreibung der Bits einer Umgebungsvariablen, sodass diese leicht interpretierbar sind.
- Für CAN-Frames werden über die Definition von **Botschaften** eindeutige Namen vergeben. Dabei werden unter anderem das Frame-Format, der Identifier sowie die Datenlänge angegeben.
- **Signale** sind für die Darstellung von Datenbytes der Botschaften ausgelegt. So kann der Entwickler sehr performant einzelne Bits über den definierten Signalnamen abfragen, ohne dabei die exakte Bitposition zu kennen oder über Shift-Operationen[2] den Bit-Wert auszuwerten. Für die Verwendung von Signalen müssen diese einer Botschaft zugewiesen sein.

[1] Es gibt an, in welcher Reihenfolge die Daten im Speicher liegen – MSB oder LSB
[2] Verschieben von einzelnen Bits im Binärzahlensystem. Eine Verschiebung von einem Bit um k nach links (1 << k) entspricht einer Multiplikation von 2^k.

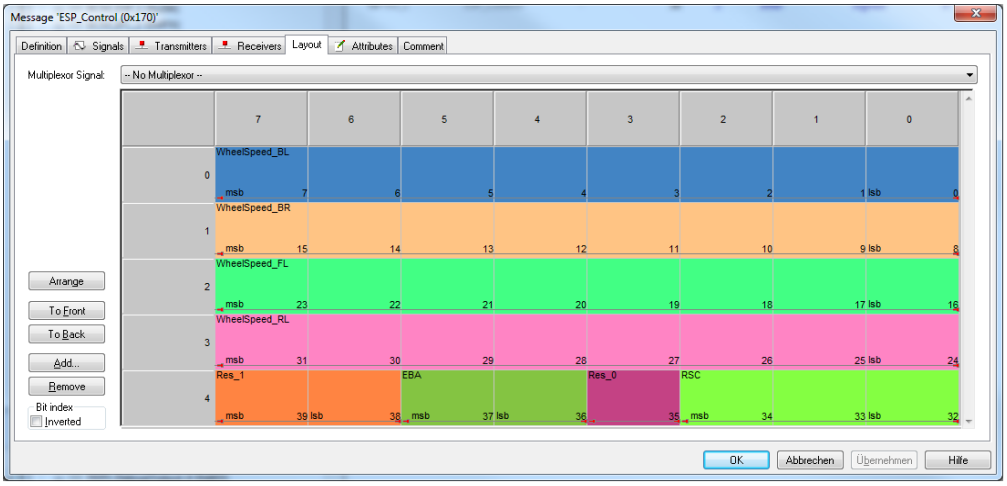

Abbildung 2.1: Visuelle Darstellung der Zuordnung von Signalen zu einer Botschaft innerhalb der Datenbank.

- Durch das Anlegen von **Steuergeräten** kann der Nachrichtenfluss definiert werden. Signale und Botschaften können mit Steuergeräten verknüpft werden, um die Sende- (Tx) und Empfangsrichtung (Rx) anzugeben.

Programmiersprache CAPL

Mithilfe der CANoe-eigenen Programmiersprache CAPL „Communication Access Programming Language“, welche sich an die Hochsprache C anlehnt, ist es möglich, individuelle Anwendungen zu programmieren. Hierbei können mit den CAPL-Knoten komplexe Steuergeräte und Logiken nachgebildet werden [Lobmeyer und Marktl 2014 a]. Bei CAPL handelt es sich jedoch, anders als bei einer prozeduralen Programmiersprache, wie es bei C der Fall ist, um eine ereignisgesteuerte Programmiersprache. Auf jedes auftretende Ereignis wird eine gleichwertige Funktion aufgerufen und ausgeführt. Der Aufruf des jeweiligen Ereignisses ist vergleichbar mit vielen einzelnen Interrupt-Service-Routinen (ISR) in C, die alle dieselbe Priorisierungsstufe haben und durch das jeweils auftretende Ereignis aufgerufen werden. Die ISRs sind nicht zeitkritisch,

weswegen nur sehr eingeschränkt auf die Ausführungszeit der Funktionen geachtet werden muss. So führt CANoe aus Modellsicht alle Aktionen zum Zeitpunkt des auftretenden Ereignisses gleichzeitig aus, also beliebig schnell. Gesendete und empfangene Botschaften enthalten über eine Echtzeituhr einen Zeitstempel, was zu einer sicheren Arbitrierung beim Senden führt [Lobmeyer und Marktl 2014 b].

Tabelle 2.1: Vor- und Nachteile von CAPL gegenüber der Hochsprache C [Lobmeyer und Marktl 2014 b]

Vorteile	Nachteile
Ereignisgesteuert – dadurch sehr flexibel und einfach zu programmieren. Es wird die Modellvorstellung verwendet, dass CAPL-Programme beliebig schnell ablaufen.	Debuggen nicht im realen Einsatz (mit angeschlossenem ECU) möglich.
Durch die Nichtverwendung von dynamischer Speicherverwaltung, ist CAPL sehr robust. Memoryleaks oder Speicherüberlauf sind somit ausgeschlossen.	Unterstützt keine Zeigerarithmetik und damit auch keine dynamische Speicherverwaltung. So werden alle Variablen statisch angelegt und der Speicherplatz erst nach Programmende wieder freigegeben, was erhöhten Speicherplatzes benötigt.
Verwendung von assoziativen Arrays[3]	Nur fest definierte „Defines“ können verwendet werden. Bedingtes Kompilieren somit nur bedingt unterstützt.
Einfache Abfrage von CAN-Nachrichten über fest definierte Ereignisse/Funktionen – bedingt durch die eingebunden Datenbanken.	
Optimal für den Einsatz im Automotive-Bereich angepasst	

[3] In anderen Programmiersprachen entsprechen assoziative Arrays Containern oder Maps, welche für die dynamische Speicherverwaltung zuständig sind. Effiziente Hashtabellen sind dabei für die Speicherverwaltung zuständig und so muss die Größe der Daten vor dem Abspeichern nicht bekannt sein [Lobmeyer und Marktl 2014 c].

Grafische Benutzeroberflächen

Der CANoe-eigene Paneleditor ermöglicht das Erstellen von grafischen Benutzeroberflächen innerhalb von CANoe. Hierbei steht dem Entwickler eine Vielzahl von grafischen Werkzeugen zur Verfügung, die strukturierte und übersichtliche grafische Benutzeroberflächen ermöglichen. Die integrierte Toolbox ist optimal an die Bedürfnisse der Automobilbranche angepasst. Umgebungs- oder Systemvariablen können direkt eingebunden und über einen CAPL-Knoten mit Logik versehen werden. Interaktive Elemente können ebenfalls eingebunden und mit Intelligenz über CAPL versorgt werden.

Des Weiteren stehen dem Entwickler weitere grafische Elemente zur Verfügung, welche die Auswertung und Analyse einer Buskommunikation ermöglichen:

- Trace-Fenster zur Anzeige der Buskommunikation
- Busstatistik
- Grafische Anzeige von Signalen
- Darstellung von Umgebungsvariablen
- Zeitliche Abhängigkeiten von Zustandsänderungen von Botschaften und Signale visuell darstellen

Diagnosebeschreibung über das CANdela-File

CANoe bietet die Möglichkeit, über Diagnosebefehle aktive Informationen vom Steuergerät anzufordern oder Fahrzeugparameter anzupassen. Die Aufgaben der Diagnosebefehle werden in Abschnitt 2.3.3 näher erläutert. Das Spezifikationswerkzeug für die formale Beschreibung der Fahrzeug-Steuergeräte-Diagnosebefehle lautet „CANdelaStudio“ [CANdela 2019]. Die erzeugten Diagnosebeschreibungen – CANdela-Files – können in CANoe integriert werden. Dabei findet eine Unterstützung von allen gängigen Diagnoseprotokollen wie z. B. UDS, KWP2000 oder ODB statt. Die

Diagnosebefehle unterliegen dabei einer standardisierten Definition, wie der ISO 14230 für KWP2000 [ISO/DIS 14230 KWP2000] oder der ISO 14229 für die UDS-Diagnoseprotokolle [ISO 14229 UDS].

2.2 Restbussimulation

Das Ziel einer Restbussimulation im Automotive-Bereich ist es, eine funktionsfähige Testumgebung während der Entwicklungsphase eines Steuergeräts oder eines kompletten Netzwerks zu haben. So wird mit einer Restbussimulation einem Steuergerät die Infrastruktur für die Kommunikation mit anderen Netzwerkteilnehmern softwaretechnisch zur Verfügung gestellt [Albrecht und Decker 2012]. Sie erlaubt dem Entwickler, zur Laufzeit des Entwicklungsprozesses seine implementierte Steuergerätesoftware ausreichend zu testen, um so Fehler frühzeitig zu erkennen.

Der Entwicklungsprozess für die Restbussimulation wird in drei Phasen (Abbildung 2.2) aufgeteilt.

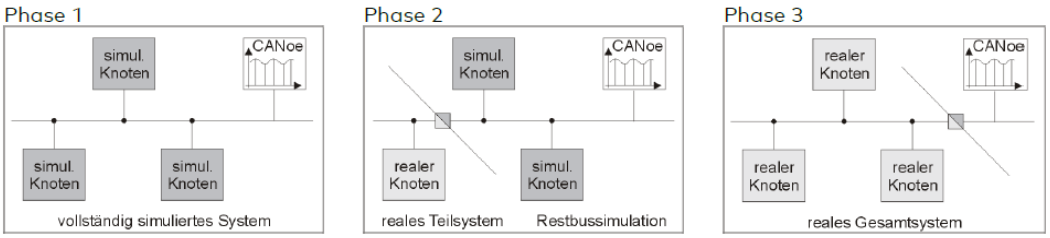

Abbildung 2.2: Entwicklungsprozess einer Restbussimulation mit CANoe [CANoe 2018, S. 20]

- **Phase 1**

  Alle relevanten Steuergeräte eines Systems werden softwaretechnisch mittels z. B. CANoe nachgebildet und interagieren miteinander über das entsprechende Bus-Protokoll wie CAN, LIN, FleyRay oder MOST.

- **Phase 2**

  Über eine externe Schnittstelle kann ein realer Bus mit angeschlossenem Steuergerät eingebunden werden. Steuergeräte, welche für die

Kommunikation mit dem realen Steuergerät zuständig sind, werden weiterhin softwareseitig betrieben. Die Kommunikation findet wieder über das entsprechende Bus-Protokoll statt. In dieser Phase wird dem reellen ECU über die Restbussimulation ein funktionsfähiges Netzwerk zur Verfügung gestellt.

- **Phase 3**
  Alle Netzwerkknoten des Systems werden an den realen Bus angeschlossen. In dieser Phase wird z. B. CANoe für die Analyse und Auswertung der Buskommunikation eingesetzt.

Bei den simulierten Knoten des virtuellen Teilsystems kann es sich im Automotive-Bereich um Steuergeräte eines Fahrzeugs handeln, welche softwaretechnisch umgesetzt werden. Aber auch Hilfsknoten, die es ermöglichen, Softwaretests in Echtzeit durchzuführen und die als Debug-Werkzeug fungieren, sind realisierbar. Für nicht sequentielle Kommunikationsabläufe können zudem grafische Oberflächen eingebunden werden, welche über die simulierten Netzwerkknoten ihre Intelligenz und Logik erhalten. Dem Anwender wird damit die Möglichkeit gegeben aktiv in die Buskommunikation einzugreifen. So kann z. B. während der Laufzeit über grafische Schalter die Bremse softwareseitig betätigt werden, um das Starten eines Fahrzeugs zu simulieren. Über Anzeigeinstrumente können Signalzustände visualisiert werden. Für eine klare und eindeutige visuelle Darstellung der Busbotschaften oder Signale dienen Datenbanken, welche in Kapitel 2.1 beschrieben sind.

2.3 Bussysteme im Automotive-Bereich

Bei einem Bussystem werden Daten zwischen einzelnen Teilnehmern innerhalb eines Netzwerks ausgetauscht. Die Kommunikation wird durch die beiden untersten Schichten (Schicht 1 und 2) des ISO/OSI-Referenzmodells beschrieben und verläuft über einen gemeinsamen Übertragungskanal. Schicht 1 (Physical Layer) definiert die Signalpegel, Bitlänge sowie elektrische und

mechanische Schnittstellen zum Übertragungsmedium. Schicht 2 (Data Link Layer) regelt unter anderem den Buszugriff der einzelnen Teilnehmer um eine kollisionsfreie Übertragung zu ermöglichen [KUNBUS 2019].

In heutigen Kraftfahrzeugen kommen unterschiedliche Bussysteme wie LIN, CAN, MOST, FleyRay, uvm. (siehe Abbildung 2.3) zum Einsatz. Der Datenaustausch zwischen den unterschiedlichen Bussystemen und somit Übertragungsprotokollen erfolgt über Gateways. Jedes dieser Protokolle erfüllt optimal spezielle Systemanforderungen wie Kosten, Datenrate, Echtzeitfähigkeit oder Fehlersicherheit, wodurch die hohe Anzahl an Bussystemen innerhalb eines einzigen Fahrzeugs zustande kommt [Zimmermann und Schmittgall 2018, S. 2 f.]. Da für diese Arbeit lediglich der CAN-Bus und darauf aufbauende Übertragungsprotokolle notwendig sind, findet eine Beschreibung der anderen Bussysteme im Rahmen dieser Arbeit nicht statt.

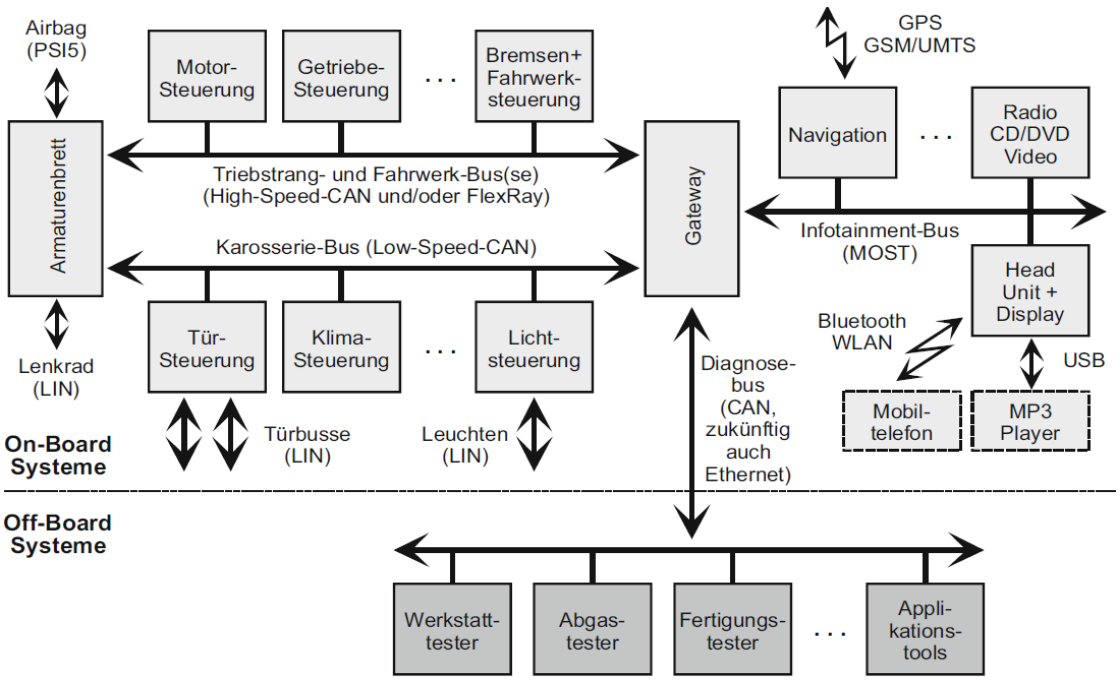

Abbildung 2.3: Bussysteme im modernen Fahrzeug [Zimmermann und Schmittgall 2014, S. 2]

2.3.1 Controller Area Network – CAN

Das CAN-Übertragungsprotokoll ist das am häufigsten eingesetzte Bussystem im automobilen Bereich [Borgeest 2014, S. 92]. Es findet sowohl Anwendung für Low-Speed- (125 kBits/s) als auch für High-Speed- (1 MBits/s) Funktionalitäten. Die Kommunikation des CAN-Protokolls läuft auf den unteren beiden Schichten des ISO/OSI-Referenzmodells (Tabelle 2.2) ab und wird durch die ISO 11898 beschrieben. Entwickelt wurde das Bussystem 1983 von der Firma Bosch, um eine Reduktion der Kabelbäume zu erlangen. Durch sein differentielles Signal und die mehrfache Absicherung bei der Datenübertragung ist eine sichere Kommunikation in einem störbehafteten Umfeld gewährleistet. Das heutige Anwendungsgebiet geht über die Steuergerätevernetzung in Kraftfahrzeugen hinaus und erstreckt sich von medizinischen Geräten über Fahrstuhlsteuerungen bis hin zur Automatisierungstechnik.

Tabelle 2.2: ISO/OSI-Referenzmodell des CAN-Bus

Schichten	ISO-Norm	Beschreibung
2	ISO 11898 -1	**Data Link Layer**, Spezifikation nach CAN 2.0A und CAN 2.0B
1	ISO 11898 - 2, 5 , 6 ISO 11898 - 3	**Physical Layer** für … High Speed CAN und … Low Speed CAN

Aufbau eines CAN-Frames

Ein Datenrahmen einer CAN-Nachricht umfasst mehrere Felder, welche die unterschiedlichsten Funktionalitäten zugewiesen bekommen (siehe Abbildung 2.4). Folgend werden die Funktionen der einzelnen Felder eines CAN-Frames erläutert [CANBUS 2018]:

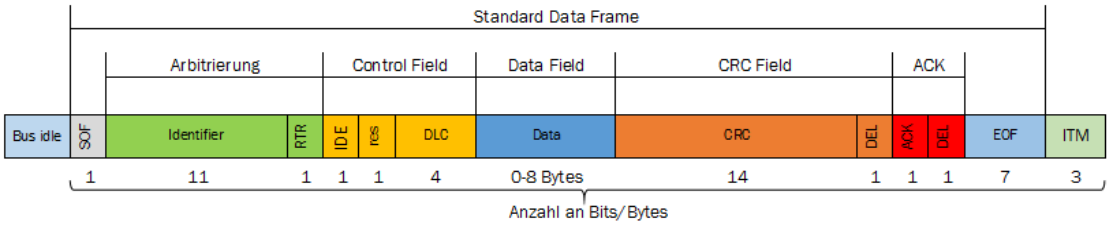

Abbildung 2.4: Standard Daten Frame

- **Start of Frame (SOF):** Durch einen Flankenwechsel von rezessiv (Bus-Idle) auf dominant wird eine Netzwerk-Synchronisation durchgeführt und die Datenübertragung beginnt.
- **Identifier (ID):** Über die ID ist jeder Netzwerkknoten in einem Netzwerk eindeutig identifizierbar. Zudem wird der Identifier für die Priorisierung und somit für den Buszugriff benötigt.
- **Remote Transmisson Request (RTR-Bit):** Dieses Bit zeigt dem Empfänger-Knoten den Frame-Typ an. Die unterschiedlichen Typen werden nachfolgend genauer beschrieben.
- **Identifier Extension (IDE):** Für eine Unterscheidung zwischen dem Standard-CAN-Format (11-Bit ID) und dem Extended-CAN-Format wird das IDE-Bit benötigt. Bei einem rezessiven Bit handelt es sich um ein Extended-Format.
- **Data Length Code (DLC):** Gibt dem Netzwerkknoten Auskunft über die Anzahl der Nutzbytes (Datenfeldgröße).
- **Data Field:** Dieses Feld, welches eine maximale Größe von 8 Bytes hat, beinhaltet die Nutzdaten.
- **Cyclic Redundancy Checks (CRC):** Durch eine Prüfsummenberechnung von Sende- und Empfänger-Knoten können Übertragungsfehler erkannt werden. Das CRC-Feld enthält die vom Sende-Knoten berechnete Prüfsumme.

- **Acknowledgement (ACK):** Ausgehend von der Prüfsumme quittiert der Empfänger bei einer korrekten Übertragung mit einem dominanten Bit. Übertragen wird das ACK-Bit immer rezessiv.
- **End of Frame (EOF):** Sieben rezessive Bits beenden die Datenübertragung.
- **Intermission (ITM):** Gibt die minimale Bit-Anzahl an, um aufeinanderfolgende Frames zu trennen – mindestens drei Bits werden jedoch benötigt.
- **IDLE:** Will nach Ablauf der ITM kein weiterer Netzwerkknoten einen Frame senden, so bleibt der Bus in seinem Ruhezustand.

Bestandteile eines CAN-Knotens

Ein CAN-Knoten besteht aus drei Einheiten:

- CAN-Transceiver
- CAN-Controller
- Host (Mikrocontroller)

Der **CAN-Transceiver** ist dabei die Schnittstelle zwischen dem CAN-Controller und dem physikalischen Bus. Er ist für die Transformation zwischen dem auf der einen Seite differentiellen Bussignal und auf der anderen Seite seriellen Controllersignal zuständig. Die Aufgaben des **CAN-Controllers** bestehen darin, einen einkommenden Frame nach der Protokollspezifikation ISO 11898 zu behandeln und das Rahmenformat nach dem CAN-Protokoll zu erfüllen. Dies bedeutet unter anderem das Berechnen und Kontrollieren der CRC-Summe, das Setzen des ACK-Bits sowie das Priorisieren der Frames über den Identifier. Der **Host** (Mikrocontroller) gibt dem Netzwerkknoten seine Adresse, über die er eindeutig in einem CAN-Netzwerk identifizierbar ist. Die ID beschreibt außerdem die Priorität des Netzwerkknotens. Je kleiner die ID, desto höher ist die Priorität [CANBUS 2018].

Frame-Typen

- **Data Frame (dominantes RTR-Bit)**
  Der Informationserzeuger (Sende-Knoten) ergreift die Initiative und versendet einen Frame via Broadcast über den CAN-Bus. Zusätzlich gibt es eine Unterscheidung über das IDE-Bit zwischen Standard- und Extended-Frame. Der Identifier umfasst beim Standard-Frame elf Bits und beim Extended-Frame 29 Bits, was eine höhere Anzahl an Netzwerkknoten ermöglicht.
- **Remote Frame (rezessives RTR-Bit)**
  Ein Netzwerkknoten fordert Informationen durch das Versenden eines Remote-Frames, von einem anderen Netzwerkknoten an. Der Frame enthält kein Datenfeld.
- **Error Frame**
  Der Error Frame weist wesentliche Unterschiede im Vergleich zu dem Data und Remote Frame auf. Es wird nur zwischen den beiden Feldern „Error Flag“ und „Error Delimiter“ unterschieden. Erkennt ein Busteilnehmer einen Busfehler, unterbricht er die Übertragung und sendet einen Error Frame, welcher aus sechs dominaten Bits besteht. Werden mehr als fünf homogene Bits gesendet, so wird der Sicherheitsmechanismus „Bit-Stuffing“ gebrochen. Dieser Mechanismus sendet nach fünf homogenen Bits ein inverses Bit. Die anderen Teilnehmer erkennen den „Bit-Stuffing“-Fehler und senden ebenfalls ein Error Frame. Die bis zu zwölf aufeinanderfolgenden dominanten Bits werden durch den „Error-Delimiter“ ergänzt. Dieser setzt sich aus acht rezessiven Bits zusammen und vervollständigt den Error Frame (siehe Abbildung 2.5). Nach den drei Intermission-Bits (ITM) kehren alle Busknoten in ihren Idle-Zustand zurück.

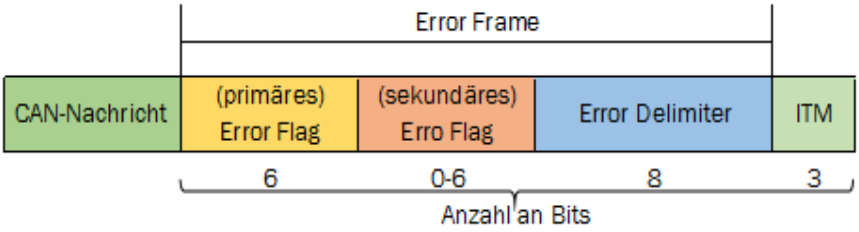

Abbildung 2.5: Beschreibung eines CAN Error-Frames [CANBUS 2018]

Kommunikationsprinzip

Um die Echtzeitfähigkeit des CAN-Busses zu gewährleisten, kommt für den CAN das CSMA/CR-Verfahren (Carrier Sense Multiple Access with Collision Resolution) zum Einsatz, was eine kollisionsfreie Übertragung ermöglicht. Sollten zwei oder mehrere Knoten gleichzeitig versuchen auf den Bus zuzugreifen wird über die Arbitrierung die Zugriffsreihenfolge geregelt. Botschaften mit höherer Priorität (kleinerer Identifier) setzen den Sendevorgang fort, Sender mit niederpriorisierten Botschaften hingegen stellen ihn sofort ein [Zimmermann und Schmittgall 2014, S. 28].

2.3.2 SAE J1939 Protokoll

Das auf dem CAN basierende Protokoll findet sein Einsatzgebiet zum Großteil im Nutzfahrzeugbereich. Es soll damit eine gemeinsame Basis für den Datenaustausch zwischen Steuergeräten unterschiedlicher Hersteller geschaffen werden. Die Spezifikation erfolgte durch die amerikanische „Society of Automotive Engineers SAE“ [J1939 2018]. Grundlage des Protokolls ist die CAN Spezifikation (ISO 11898). Sie regelt die Datenübertragung auf den beiden unteren Schichten des ISO/OSI-Referenzmodells, bei dem der Aufbau ebenfalls in Hardware erfolgt. Die Netzwerk-, Transport- und Applikationsschichten werden softwareseitig umgesetzt und sind in der SAE J1939 spezifiziert (siehe Abbildung 2.6).

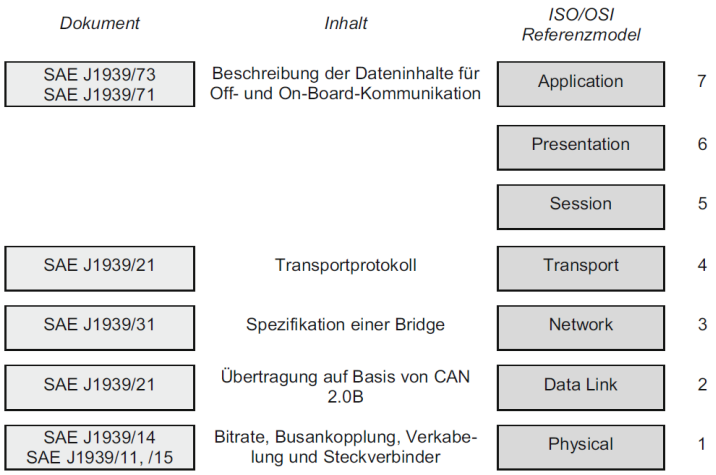

Abbildung 2.6: Protokollstapel nach SAE J1939 [Zimmermann und Schmittgall 2014, S. 178]

Dadurch ergeben sich folgende Erweiterungen und Eigenheiten [J1939 2018]:

- Ein Datenfeld kann acht Bytes mithilfe der Parametergruppe überschreiten. Da jedoch CAN als physikalisches Übertragungsmedium eingesetzt wird, werden sie in einzelne Pakete segmentiert und über den Bus versendet. Über ein Transportprotokoll (ähnlich TCP) werden die einzelnen Fragmente beim Empfänger wieder zusammengesetzt. Dafür ist im J1939 Standard ein Regelwerk definiert.
- Punkt-zu-Punkt-Adressierung (Knotenadressierung) sowie globale Adressierung (Botschaftsadressierung).
- Definiert eigene Diagnoseschnittstelle (J1939 Diagnose) sowie ein Standard-Diagnosestecker: Anders als bei der UDS-Diagnose, bei der die Dienste aktiv über ein Softwarewerkzeug angestoßen werden müssen, senden J1939-ECUs auch selbständig Diagnosenachrichten während des Standardbetriebs.
- Änderung der Bustopologie zur Laufzeit möglich (z. B. Kommunikation zwischen Traktor und Anbaugerät, welche sich zur Laufzeit ändern können).

Struktur für einen J1939-Identifier

Der Identifier einer J1939 Botschaft ist in drei Segmente (Priorität, Parametergruppe und Quelladresse) unterteilt (Abbildung 2.7). Nachfolgend werden die Aufgaben der einzelnen Segmente erläutert [Zimmermann und Schmittgall 2014, S. 179 f.].

- **Priorität:** Die ersten drei Bits geben die Priorität der Nachricht an. Dabei gilt, je kleiner der Identifier desto größer ist die Priorität der Nachricht.
- **Parametergruppe:** Sie gibt Aufschluss über die Art der Nachricht. So wird das PDU 1-Format für Nachrichten genutzt, bei welchen nur ein einzelnes Steuergerät angesprochen werden soll (Punkt-zu-Punkt-Verbindung). Die Destination-Adresse gibt dabei die Adresse des Zielsteuergeräts an. Das PDU 2-Format wird hingegen für nachrichtenorientierte Botschaften verwendet, bei welchen alle Netzwerkknoten die versendeten Nachrichten enthalten. Die ersten beiden Bits der Parametergruppe sind für zukünftige Erweiterungen vorgesehen.
- **Quelladresse:** Diese acht Bits geben die Adresse des Steuergeräts an.

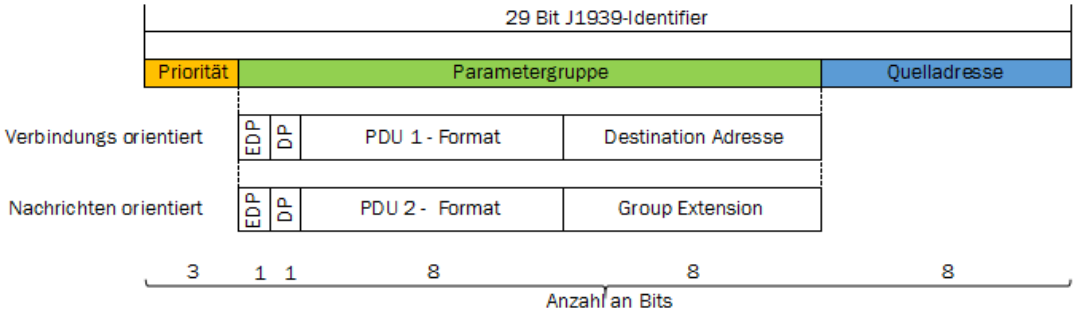

Abbildung 2.7: J1939-Identifier

2.3.3 Diagnoseprotokoll

Die Funktionalitäten des Protokolls bauen auf dem für K-Line-Verbindungen definierten KWP 2000-Diagnoseprotokoll auf und ermöglichen eine

Implementierung für CAN-Bussysteme. Da für diese Arbeit K-Line nicht relevant ist, wird darauf nicht weiter eingegangen.

Bei der Diagnoseschnittstelle wird zwischen On-Board-Diagnose und Off-Board-Diagnose unterschieden. Dabei findet eine Interaktion auf der Applikationsschicht (Schicht 7) des ISO/OSI-Referenzmodells zwischen dem Steuergerät und dem Fahrzeugführer, Entwickler oder Tester statt.

On-Board-Diagnose

Diese Diagnoseschnittstelle ermöglicht ein standardisiertes Aufzeichnen von Systemfehlern jeglicher Art, welche durch entsprechende Sensoren überwacht werden. Beispielsweise findet zur Laufzeit eine Überwachung und Aufzeichnung der abgasrelevanten Systeme statt, welche den gesetzlichen Regelungen entsprechen [Zimmermann und Schmittgall 2014, S. 189]. Je nach Fehlerfall wird unterschiedlich reagiert. So erfolgt auf fahrzeugkritische Fehler eine Warnmeldung an den Fahrzeugführer und das Fahrzeug wird ggf. in ein Notprogramm versetzt. Weniger kritische Systemfehler werden dauerhaft über genormte Fehlercodes (DTC) in dem entsprechenden Steuergerät gespeichert und können über die Diagnoseschnittstelle in einer Kfz-Werkstatt ausgelesen und analysiert werden.

Off-Board-Diagnose

Das Kommunikationsmodell für die Off-Board-Diagnose sieht vor, dass die gesamte Kommunikation vom Tester gesteuert wird und dieser die komplette Kontrolle über den Datenfluss besitzt [Zimmermann und Schmittgall 2014, S. 192]. Der Kommunikationsfluss wird in Abbildung 2.8 dargestellt.

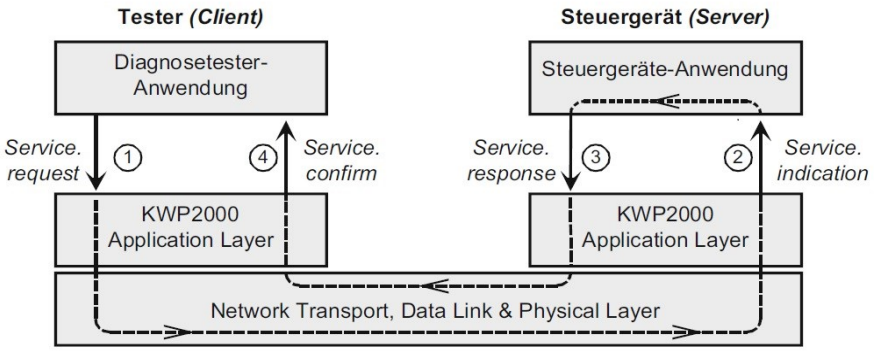

Abbildung 2.8: Kommunikationsmodell der KWP 2000-Diagnose [Zimmermann und Schmittgall 2014, S. 192]

1) Der Tester sendet eine Diagnose-Anfrage (Service-Request) über den Kommunikationsbus an das Steuergerät.
2) Über die Applikationsebene wird das Steuergerät über die Anfrage des Testers informiert (Indication).
3) Auf die empfangene Anfrage antwortet das Steuergerät und sendet eine Antwort (Response) an den Tester.
4) Die Applikationsebene übergibt die Antwort (Confirm) der ursprünglichen Anfrage an den Tester.

Die Off-Board Diagnose erlaubt dem Entwickler/Tester den Zugriff auf das Steuergerät. Dies ermöglicht die Fahrzeugparameter zu verändern, den Fehlerspeicher auszulesen, zu löschen oder ein Steuergerät zu „flashen[4]“ [Zimmermann und Schmittgall 2014, S. 433 f.]. Der Zugriff wird über die Diagnosesitzung geregelt. Je nach Entwicklungsstatus werden unterschiedliche Sitzungen mit unterschiedlichen Zugriffstiefen ermöglicht. Für die Diagnosesitzungen werden Sicherheitsschlüssel benötigt, welche über ein „Seed & Key“-Verfahren berechnet werden. Die Algorithmen für die Berechnung sind geheim und liegen nur dem Entwickler und im Steuergerät vor. Somit ist ein Zugriff durch Dritte auf das ECU nicht möglich. Der Ablauf wird durch ein

[4] Übertragen und Speichern von Software und Daten in den dauerhaften Speicher (Flash-Speicher) des Steuergeräts.

Sequenzdiagramm in Abbildung 2.9 beschrieben. Das Programmiergerät stellt dabei den Entwickler/Tester dar.

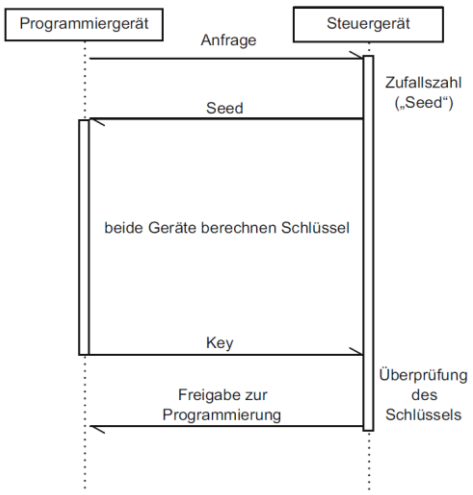

Abbildung 2.9: "Seed & Key"-Verfahren [Borgeest 2014, S. 276].

2.4 Keyless

Moderne Fahrzeuge haben die Möglichkeit eines schlüssellosen Komfortzugangs. Das bedeutet, dass ein Fahrzeug ohne aktive Schlüsselbenutzung ent- und verriegelt werden kann. Der Fahrzeugschlüssel mit der passenden ID muss sich lediglich in Fahrzeugnähe befinden (siehe Abbildung 2.10). Die Reichweite hängt dabei von den unterschiedlichen Herstellerspezifikationen ab. Ein guter Mittelwert ist dabei ein Radius von ca. 1,5 m. Der Startvorgang erfolgt wiederum schlüssellos, hierfür muss sich der Schlüssel jedoch im Fahrzeuginnenraum befinden [HELLA 2019]. Die Positionsbestimmung des Fahrzeugschlüssels erfolgt über verschiedene LF-Antennen, welche sich hinter der Fahrzeugkarosserie und im Innenraum befinden, um ein sicheres Ent- und Verriegeln sowie Starten zu ermöglichen. Auf die Antenne im Fahrzeuginnenraum (Transponder-Antenne) wird später eingegangen.

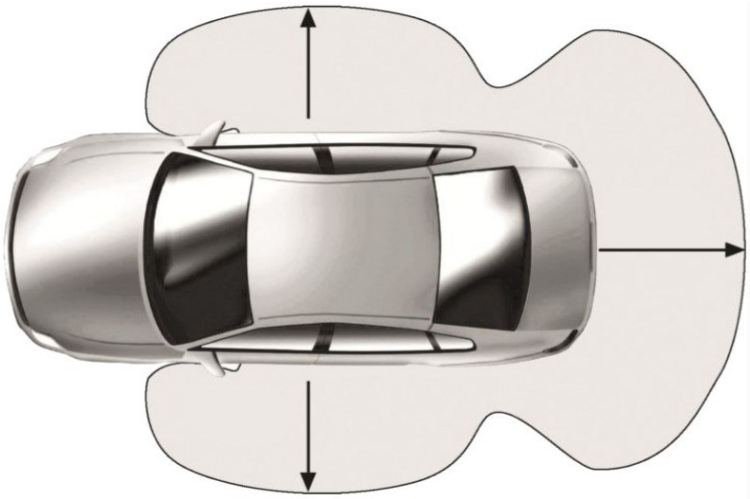

Abbildung 2.10: Funktionaler Bereich des Keyless-Schlüssels für die Ent- und Verriegelung eines Fahrzeugs [HELLA 2019].

Die Keyless-Funktionen werden vom Komfortsteuergerät übernommen, welches noch weitere Funktionen implementiert hat:

- Zentralverriegelung
- Komfortöffnen und -schließen der Fenster
- Wegfahrsperre

Ent- und Verriegelungsvorgang sowie Startvorgang eines Fahrzeugs mit integriertem Keyless-System

Getriggert durch einen Sensor[5] an einem Türgriff senden die LF-Antennen[6] elektromagnetische Felder aus, um die Position des Schlüssels zu bestimmen. Wird der Schlüssel mit der passenden ID in dem spezifizierten Bereich des Fahrzeugs erkannt, wird ein Verify für das Entriegeln gegeben. Die Schlüsselantwort, und somit das mechanische Entriegeln benötigt eine Zeitspanne von < 100 ms. Diese schnelle Reaktionszeit ist notwendig, um dem

[5] Kapazitive oder mechanische Ausführung je nach Fahrzeugklasse möglich
[6] Antennen, welche im niederfrequenten Frequenzband bis zu 125 KHz senden

Fahrzeugführer optimalen Komfort zu bieten. Analog zum Entriegelungsvorgang läuft der Startvorgang ab. Durch das Betätigen des Start-Stopptasters sowie der Bremse werden ebenfalls die LF-Antennen des Fahrzeugs über das Steuergerät getriggert, um die Schlüsselposition zu bestimmen. Befindet sich der Fahrzeugschlüssel mit der passenden ID im Fahrzeuginnenraum, so lässt sich das Lenkrad entriegeln und das Fahrzeug starten. Ist dies nicht der Fall, wird eine Lenkrad- und Startfreigabe verhindert. Durch die aktive Benutzung der Schlüsseltasten kann zudem ein manuelles Ver- und Entriegeln durchgeführt werden. Dies ist zum Beispiel für die Fahrzeugfindung in einem Parkhaus nützlich.

Um einen Startvorgang ohne ausreichende Batteriespannung gewährleisten zu können, kommt hierfür die Transponderantenne in der Mittelkonsole des Fahrzeugs zum Einsatz. Sie versorgt die Schlüsselelektronik über ein Magnetfeld mit ausreichend Energie, um eine Startfreigabe durch den ID-Geber zu ermöglichen. Zusätzlich wird sie in der Entwicklungsphase unter anderem für das „Schlüsselanlernen“ (Kapitel 6.1) benutzt.

3 Anforderungen an die Plattform-RBS

In diesem Kapitel werden die Anforderungen aufgelistet, welche an die Plattform-Restbussimulation gestellt werden. Dabei wird zwischen den funktionalen und den nichtfunktionalen Anforderungen unterschieden.

3.1 Funktionale Anforderungen

Die funktionalen Anforderungen beinhalten die elementaren Funktionen, welche für die Plattformsimulation unabdingbar sind. Die Abschnitte

- Kundenabdeckung der Konfiguration
- dynamische Kundenauswahl
- Unterstützung des CAN- und J1939-Protokoll

werden nachfolgend beschrieben und beinhalten die wichtigsten Anforderungen an das System.

Kundenabdeckung der Konfiguration

Die Plattformsimulationsumgebung soll alle OEM-Steuergeräte, welche die Plattformsoftware implementiert bekommen, abdecken. Dabei sind die folgenden Module in CANoe für die verschiedenen OEMs notwendig:

- Datenbanken (*.dbc)
- Netzwerkknoten (*.can)
- Grafische Benutzeroberflächen (*.xvp)
- CANdela-Dateien (*.cdd oder *.odx)

Die vier aufgezählten Module werden mindestens benötigt, um die unterschiedlichen OEM-Steuergeräte an die Restbussimulation zu binden und grundlegende Funktionstests durchführen zu können.

Dynamische Kundenauswahl

Um auf die herstellerspezifischen Parameter und Module richtig reagieren zu können, muss dem Benutzer klar sein, welches Steuergerät er verwendet und testet. Dabei soll dynamisch auf die herstellerspezifischen Botschaften und Parameter reagiert werden.

Die Kundenauswahl der OEMs soll dynamisch umgesetzt werden. Das bedeutet, dass die Plattform-Steuergeräte zur Laufzeit identifiziert und entsprechend die Kundenparameter gesetzt werden.

Unterstützung des CAN- und J1939-Übertragungsprotkolls

Da einzelne Plattform-Kunden das CAN-Protokoll und andere das J1939-Protokoll für die Kommunikation benutzten, soll die Restbussimulation beide Protokolle unterstützen können. Die Protokolldefinition der Kunden ist in der Netzwerkdefinition der Datenbanken (Abbildung 3.1) hinterlegt und soll automatisch angepasst werden.

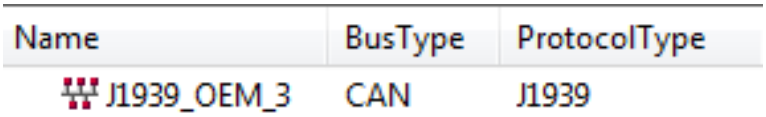

Abbildung 3.1: Protokoll-Definition (J1939-Protokoll) in der Datenbank für das Netzwerk J1939_OEM_3

3.2 Nichtfunktionale Anforderungen

Dieser Abschnitt beschreibt Anforderungen, bei denen nicht die Funktionalität der Simulationsumgebung im Vordergrund steht. Vielmehr wird bei den nichtfunktionalen Anforderungen auf Qualitätsmerkmale wie Performance, Wartbarkeit, Erweiterbarkeit und Ausgereiftheit der Restbussimulation geachtet. So soll für die Implementierung der Simulationsmodule Redundanz durch mehrmaliges Einbinden gleicher oder sehr ähnlicher Module vermieden werden. Diese Module sollen dann zusammengefasst werden, um den laufenden Verwaltungsaufwand zu reduzieren und die Übersichtlichkeit zu wahren.

Übersichtliche Simulationsumgebung

Um eine benutzerfreundliche CANoe-Konfiguration aufbauen zu können, sollen die Netzwerke, welche für eine Restbussimulation der einzelnen Kunden notwendig sind, intuitiv und klar strukturiert angelegt werden. Die folgenden Punkte sollen für eine klare Struktur der Konfiguration, insbesondere der Netzwerkaufteilung, sorgen.

- Kundenspezifische Netzwerke, Netzwerkknoten, Datenbanken, Panels oder Diagnosebeschreibungen sollen über Suffixe (z. B. das Anlegen eines Netzwerks mit „CAN_BOSCH“) einheitlich und klar strukturiert angelegt werden.
- Es soll eine klare Unterteilung für die kundespezifischen Netzwerke geben. Damit soll eine einfache Wartung für die Simulationsumgebung gegeben sein.
- Grafische Panels sollen über die Namensgebung den CAPL-Knoten klar zugeordnet werden können. Beispielsweise beinhaltet der Netzwerkknoten „SetCustomer.can“ die Logik für das „SetCustomer.xvp“-Panel. Panels, welche weniger als fünf

Funktionalitäten beinhalten, sollen zur besseren Übersichtlichkeit vermieden werden.

- Kundenspezifische Panels sollen in separaten Fenstern innerhalb CANoe anlegt werden.

Erweiterbarkeit für weitere Fahrzeughersteller/Kunden

Der Aufbau der Simulation soll so gestaltet sein, dass weitere OEMs einfach, schnell und anwenderfreundlich in die Plattform-Restbussimulation eingebunden werden können. Die Einbindung soll dabei durch die klaren, strukturellen Vorgaben der bereits integrierten Kunden vorgegeben sein und nicht von diesen abweichen.

Echtzeitfähigkeit

Die Plattformkonfiguration soll höchstens so viel Over-Head[1] besitzen, dass eine echtzeitfähige CAN-Kommunikation gewährleistet werden kann. So soll zur Simulationslaufzeit das Senden und Empfangen aller CAN-Frames gewährleistet sein, ohne dass Botschaften verloren gehen oder es zu einem Speicherüberlauf im CAN-Controller kommt, was ein Error Frame zur Folge hätte.

- Redundanter Code (CAPL) soll vermieden werden, indem er in gemeinsame Knoten zusammengefasst wird.
- Ressourcensparend entwickeln, z. B. Abfrage von einzelnen Botschaften über Namensräume.
- Deaktivieren nicht verwendeter Knoten – sorgt zusätzlich für eine übersichtliche Konfiguration.

[1] Zusätzliche Daten und Informationen zu den Nutzdaten der CANoe-Konfiguration, welche durch das Einbinden aller Plattformkunden zwangsläufig zustande kommen.

4 Konzeptentwicklung für die Plattform-Restbussimulation

Um eine übersichtliche, effiziente sowie einfach erweiterbare und bedienbare Restbussimulation aufzubauen, gibt es eine Vielzahl an Möglichkeiten. Dabei müssen Qualitätsmerkmale, wie in Kapitel 3.2 erläutert, eingehalten werden. Dieser Abschnitt stellt verschiedene Konzepte für die Umsetzung der Anforderungen dar. Dabei wird auf die verschiedenen Module, welche CANoe zur Verfügung stellt, eingegangen und die Vor- und Nachteile der jeweiligen Konzepte werden gegenübergestellt.

4.1 Aufbau der Plattform-Restbussimulation

Um die Übersichtlichkeit des simulierten Netzwerks mit den eingebunden ECUs zu gewährleisten, müssen diese möglichst kundenspezifisch aufgebaut sein. Dieser Aufbau stellt später das Grundgerüst der Simulationsumgebung dar und soll eine intuitive Benutzung sowie Erweiterung ermöglichen.

Ansatz 4.1.1 – Anlegen eines globalen CAN-Netzwerks

Anlegen eines großen CAN-Netzwerks, welches alle ECUs und Datenbanken, die in der Plattform vorkommen, enthält. Hierbei werden alle Kunden, die durch die Plattform unterstützt werden, in einem großen CAN-Netzwerk verwaltet (siehe Abbildung 4.1). Physikalisch wird über einen Bus mit dem angeschlossenen ECU kommuniziert. Zur Laufzeit sind nur die kundenspezifischen Netzwerkknoten sowie die globalen Knoten aktiviert.

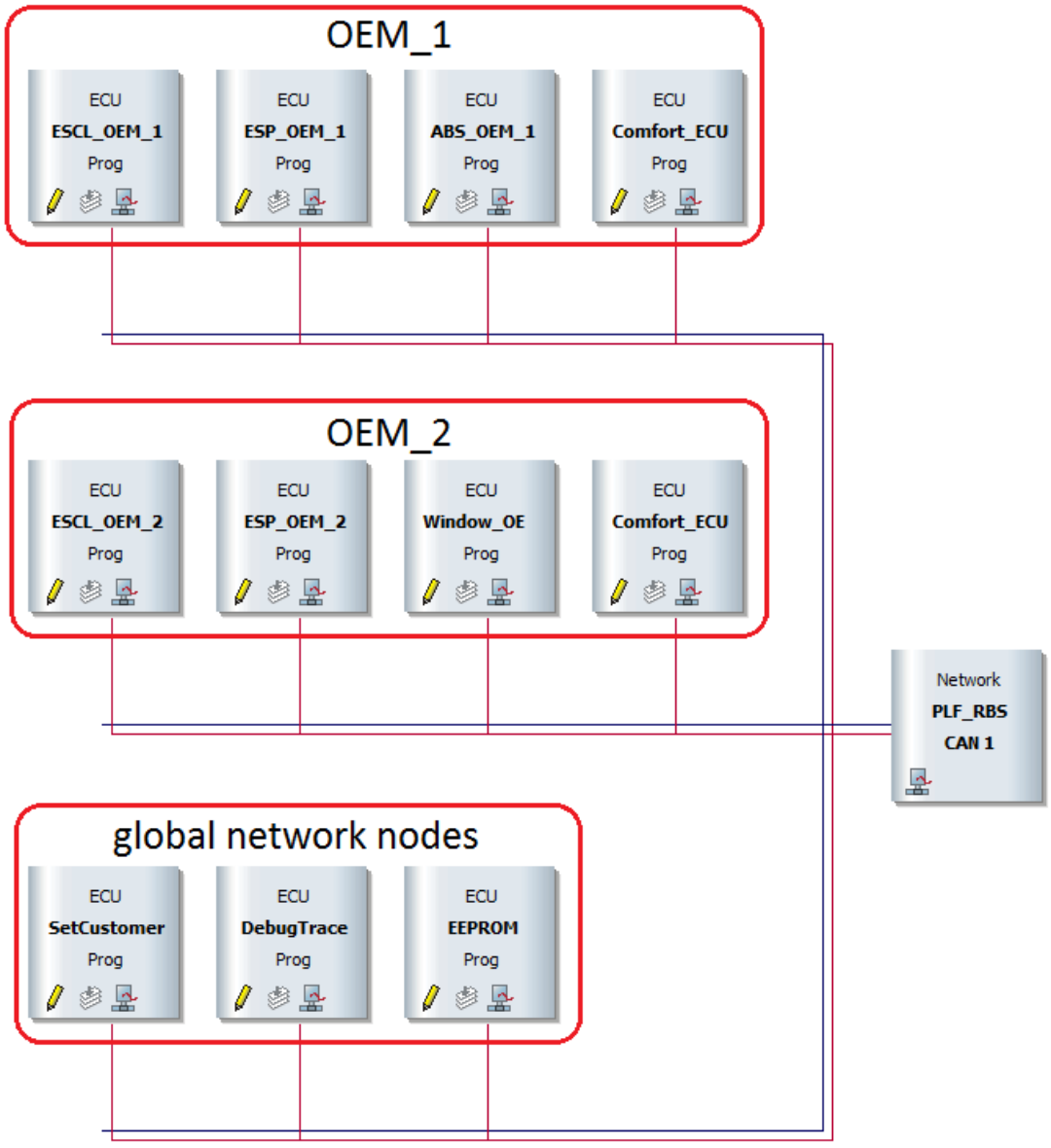

Abbildung 4.1: Alle Netzwerkknoten der Plattform werden in einem Netzwerk verwaltet

Tabelle 4.1: Vor- und Nachteile für den Ansatz 4.1.1 – Anlegen eines globalen Netzwerks

Vorteile	Neue Knoten können schnell und leicht eingebunden werden
	Kommunikation zwischen den simulierten Knoten und dem realen Steuergerät verläuft über einen gemeinsamen Kanal, welcher sich für unterschiedliche OEMs nicht ändert.
	DBC-Files werden nur einmalig eingebunden und verwaltet
Nachteile	Unübersichtliche Darstellung der simulierten Netzwerkknoten sowie der Datenbasen
	Keine eindeutige Zuordnung zwischen ECU und Kunde möglich
	Jeder simulierte Netzwerkknoten, welcher nicht zum angeschlossenen Steuergerät gehört, muss manuell verwaltet werden. Dadurch ergibt sich ein hoher Verwaltungsaufwand, welcher sehr fehleranfällig ist.
	Schwer erweiterbar
	Es kann pro erstelltem Netzwerk nur eine Diagnosebeschreibung (CANdela-File) zugewiesen werden. Aufwendige Anpassung und Einstellung beim Wechsel des Herstellersteuergeräts.

Ansatz 4.1.2 – Aufteilung in separate Kundennetzwerke

Für jeden Kunde, welcher innerhalb der CAN-Konfiguration unterstützt werden soll, wird ein separates CAN-Netzwerk angelegt (Abbildung 4.2 und Abbildung 4.3). CANoe begrenzt dabei die Anzahl der CAN-Netzwerke auf 32, was vollkommen ausreicht. Zudem soll ein CAN-Netzwerk (Abbildung 4.4) angelegt werden, welches kundenübergreifende Informationen verwaltet und redundante Module der einzelnen Kunden implementiert. Innerhalb des Netzwerks werden spezifische CAPL-Knoten und Datenbanken eingebunden.

Kundenspezifische Netzwerke

- Das kundenspezifische Netzwerk, welches nicht für die Kommunikation mit dem angeschlossenen ECU zuständig ist, wird deaktiviert (Abbildung 4.2).
- Das Netzwerk, welches für die Interaktionen mit dem physikalisch angeschlossen ECU zuständig ist, muss aktiv sein. Der Netzwerkknoten, welcher das reale ECU simuliert, wird deaktiviert (Abbildung 4.3).
- Jedem Netzwerk, das einen Restbus für einen OEM darstellt, wird eine Diagnosebeschreibung zugewiesen.
- Datenbanken, welche die kundespezifischen Daten wie Botschaften, Signale, Steuergeräte sowie Netzwerkbeschreibungen enthalten, werden den Netzwerken zugewiesen.

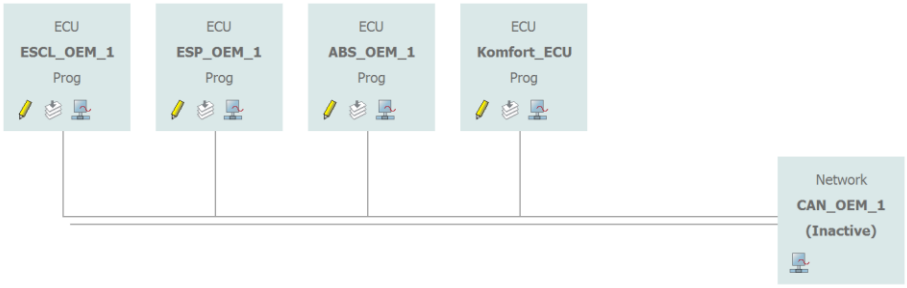

Abbildung 4.2: Inaktives Netzwerk für den ersten Plattform OEM

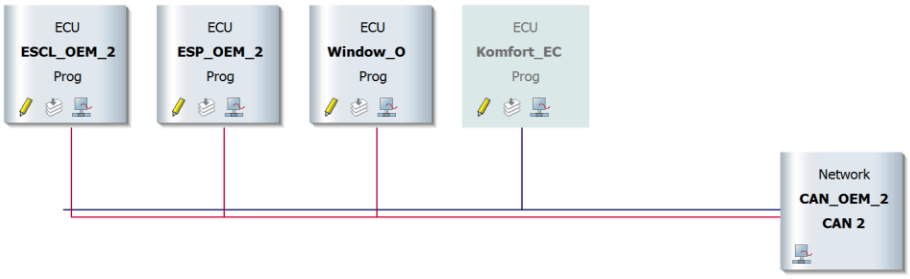

Abbildung 4.3: Aktives Netzwerk für den zweiten Plattform OEM

Kundenübergreifendes Netzwerk

- Die kundenübergreifenden Netzwerkknoten (Abbildung 4.4) bündeln den redundanten Code der Plattform-OEMs.
- Dieses Netzwerk beinhaltet globale Datenbanken, welche von allen Plattformkunden benötigt werden.
- Dieses Netzwerk ist zudem für die Zuordnung des ECU zuständig, sodass es immer aktiv sein muss.
- Separat simulierter Netzwerkkanal (CAN-Bus), für die Datenkommunikation zwischen den aktiv simulierten Netzwerken und dem realen CAN-Bus.

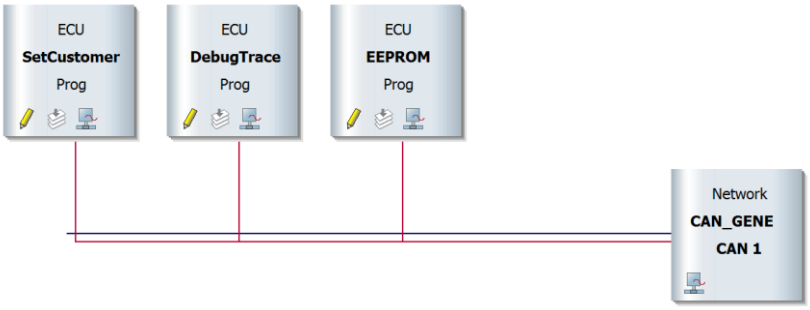

Abbildung 4.4: Globales Netzwerk, welches die Netzwerkknoten der Plattform enthält

Tabelle 4.2: Vor- und Nachteile für den Ansatz 4.1.2 – Aufteilung in separate Netzwerke

Vorteile	Übersichtliche Darstellung, da jedem Hersteller seine individuellen CAPL-Knoten zugeordnet sind
	Neue Kunden können leicht in die Plattformkonfiguration eingebunden werden
	Durch das Anlegen der herstellerübergreifenden ECUs in einem eigenen Netzwerk müssen Änderungen nur an einem zentralen Punkt vorgenommen werden
	Komplette Netzwerke einzelner Hersteller können über einen zentralen Punkt deaktiviert werden
	Jedem Netzwerk, welches einen Fahrzeughersteller repräsentiert, kann die entsprechende Diagnosebeschreibung hinzugefügt werden
Nachteile	Globale CAPL-Knoten müssen bei Kundenerweiterungen manuell angepasst werden
	Kanalzuordnung muss bei einer Kundenänderung in den Treibereinstellungen manuell angepasst werden

4.2 Dynamische Protokollanpassung

Da die Plattform-Restbussimulation die beiden Übertragungsprotokolle CAN und J1939 unterschützen soll, muss je nach ausgewähltem Hersteller zwischen den beiden genannten Protokollen unterschieden werden. Die Definition der Übertragungsprotokolle befindet sich in den Datenbanken. Über die Netzwerkattribute innerhalb der Datenbank kann den Netzwerken das Übertragungsprotokoll zugewiesen werden (siehe Abbildung 4.5). Für die Verwendung von Netzwerken, welche die Daten auf dem J1939-Protokoll übertragen, wird eine separate Lizenz von der Firma Vektor benötigt. Um die kostenpflichtige Lizenz nicht dauerhaft zu benötigen, soll das Protokoll in den Datenbanken herstellerspezifisch abgeändert werden können.

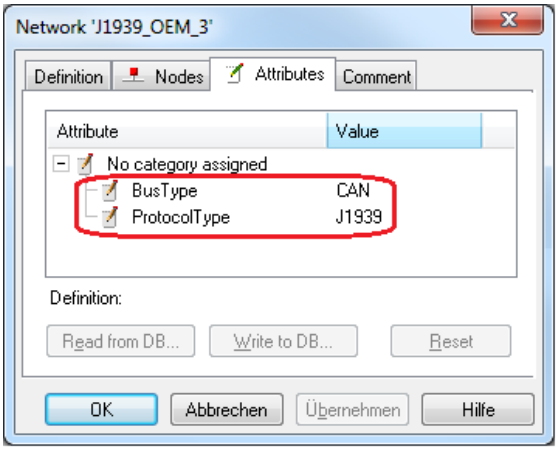

Abbildung 4.5: Netzwerkattribute für die Definition des Bus- und Protokolltyps

Ansatz 4.2.1 – Manuelle Anpassung der Übertragungsprotokolle

Die Attribute der Datenbanken werden manuell innerhalb der herstellerspezifischen Datenbanken abgeändert. Dies bedeutet, dass lediglich eine Anpassung an den Datenbanken erfolgt, welche das J1939-Protokoll unterstützen. Dabei soll das Attribut für die Beschreibung des Übertragungsprotokolls manuell innerhalb der Datenbank abgeändert werden können.

Tabelle 4.3: Vor- und Nachteile für den Ansatz 4.2.1 – Manuelle Anpassung der Übertragungsprotokolle

Vorteile	Netzwerkattribute sind übersichtlich und leicht abzuändern
	Kein zusätzlicher Implementierungsaufwand durch zusätzliche Skripte, welche die Protokolle dynamisch updaten
Nachteile	Fehleranfällig, durch manuelle Benutzereingaben innerhalb der Datenbanken
	Erhöhter Zeitaufwand um die Protokolle bei Kundenänderung/Protokolländerung manuell anzupassen

Ansatz 4.2.2 – Dynamische Anpassung der Übertragungsprotokolle

Eine extern ausführbare Datei soll die Datenbanken, welche das J1939-Protokoll benutzten, interpretieren und dynamisch an den eingestellten OEM anpassen. Über ein File-Button[1] kann die Programmdatei ausgeführt werden. Dabei ist das Attribut für die Beschreibung des Übertragungsprotokolls entsprechend anzupassen.

Als Programmiersprache wird die Hochsprache C++ verwendet. Durch das Kompilieren und Erstellen einer ausführbaren Datei hat diese Hochsprache einen entscheidenden Vorteil gegenüber der Skriptsprache Python. So wird für das Ausführen eines Python-Skripts ein Interpreter benötigt, welcher nicht allen Entwicklern zur Verfügung steht und somit ist die sichere Ausführung des Skripts nicht gewährleistet. Das erstellte ausführbare Programm durch einen C++ Kompilierer kann hingegen ohne zusätzlichen Interpreter ausgeführt werden.

Tabelle 4.4: Vor- und Nachteile für den Ansatz 4.2.2 – Dynamische Anpassung der Übertragungsprotokolle

Vorteile	Automatisches Update der Übertragungsprotokolle minimiert die Fehlerrate
	Anzahl der Datenbanken, welche das J1939-Protokoll verwenden, ist unkritisch und nicht zeitrelevant
Nachteile	Einmaliger hoher Implementierungsaufwand, um das Skript zu erstellen
	Nachfolgende Datenbanken, welche das J1939-Protokoll verwenden, müssen ebenfalls dem Skript bekannt gemacht werden, um die Protokolle automatisch upzudaten

[1] Grafisches Element innerhalb des Panel-Editors, bei dem Dateien verlinkt und ausgeführt werden können.

4.3 Dynamisches Handling der Hersteller

Um die Parameter und die kundenspezifischen Einstellungen in der CANoe-Konfiguration für das angeschlossene Steuergerät richtig setzen zu können, muss man den jeweiligen ECU-Hersteller auswählen. Folgend sind einige Parameter bzw. kundenabhängige Einstellungen aufgelistet:

- Redundante Botschaft-IDs verschiedener Kunden, welche in den kundenabhängigen Datenbanken definiert sind
- Kundenabhängige Reaktionen auf gleiche Ereignisse
- Unterschiedliche Signal-Interpretation der einzelnen Kunden

Zudem können die verschiedenen Kunden identische Botschaft-IDs besitzen. Die Deutung der Botschaft ist dabei kundenspezifisch. In diesem Fall kann über die „Qualifier“, welche auf das entsprechende Kundennetzwerk referenzieren, zwischen den identischen Botschaft-IDs unterschieden werden.

Nachfolgend werden Ansätze erläutert, um den Kunden möglichst dynamisch und performant zu ermitteln und entsprechend zu setzen. Eine Umgebungsvariable in einer Datenbank soll dabei den globalen Zugriffspunkt innerhalb der Restbussimulation darstellen. Kundenspezifische Wertetabellen, das Anpassen der spezifischen Kundennetzwerke oder der Diagnosebeschreibungen für das angeschlossene ECU müssen manuell ausgeführt werden, da Änderungen zur Simulationslaufzeit seitens CANoe nicht erlaubt sind. Dabei können die Anpassungen über Skripte automatisiert werden, was jedoch nicht Bestandteil dieser Arbeit ist.

Ansatz 4.3.1 – Manuelle Auswahl des Kunden über grafische Benutzeroberfläche

Über ein grafisches Fenster wird der Steuergeräte-Hersteller ausgewählt und dabei werden die entsprechenden RBS-Parameter gesetzt.

Tabelle 4.5: Vor- und Nachteile für den Ansatz 4.3.1 – Manuelle Auswahl des Kunden über grafische Benutzeroberfläche

Vorteile	Einfache Implementierung und Bedienung
	Über die externe Datenbank kann jeder beliebige Kunde als „Default“ ausgewählt werden. Dadurch wird der Kunde schon vor Programmstart initialisiert.
Nachteile	Manuelles Eingreifen in die RBS, dadurch fehleranfällig

Ansatz 4.3.2 – Automatische Auswahl des Kunden über die ECU-Softwarenummer

Über die Softwareversion des angeschlossenen Steuergeräts wird der entsprechende Kunde ermittelt. Um Softwareparameter abfragen zu können, muss ein Zugriff an das Steuergerät über eine Diagnosesitzung erfolgen. Dabei wird für das Abfragen der Softwarenummer ein Diagnosebefehl, welcher in der Diagnosebeschreibung des Kunden definiert ist an das angeschlossene ECU gesendet (Listing 4.1).

Listing 4.1: Auslesen der Softwarenummer für den Kunden Mahindra[2]

```
/* Define a diagnostic variable for the sw-number                    */
/*   -> the diagnostic variable are declared in the CANdela-file     */
/*      of the OEM Mahindra                                          */
diagRequest PKE_Mahindra.SystemSupplierECUSoftwareVersionNumber_Read  reqCustomer_MAHINDRA;

/* send a request command to get the sw-number of the Mahindra ECU   */
diagSendRequest(reqCustomer_MAHINDRA);
```

Wird der Befehl erfolgreich empfangen, schickt das ECU die angeforderte Antwort mit der Softwarenummer zurück. Dabei wird in der Empfangsfunktion die Softwarenummer mit einer konstanten, fest definierten Nummer verglichen und ausgewertet (Listing 4.2). Anschließend erfolgt eine Ausgabe an den Benutzer, welcher über die empfangene Softwarenummer des ECU informiert wird.

[2] Mahindra&Mahindra Limited ist ein indischer Automobilhersteller

Listing 4.2: Response-Funktion der ECU-Softwarenummer für Mahindra

```
/*-----------------------------------------------------------------------*/
/* Response of the OEM Mahindra&Mahindra for the sw-number               */
/*-----------------------------------------------------------------------*/
on diagResponse PKE_Mahindra.SystemSupplierECUSoftwareVersionNumber_Read
{
  char c_softwareCustomerNr[20];
  byte data[4096];
  long size;
  int i = 0;

  /* get the length of the response message */
  size=this.GetPrimitiveSize();
  /* copy the response message into the array */
  this.GetPrimitiveData(data, elcount(data));

  /*write the softwarenumber into the char array */
  for(i=3; i<20; i++)
  {
    c_softwareCustomerNr[i-3] = data[i];
  }

  /* check if the sw-number is equal to the firmly defined number of Mahindra */
  if(check_softwarenumber(c_softwareCustomerNr))
  {
    /*write the sw-number into the "SetCustomer-Panel" for user instructions */
    putValueToControl("SetCustomer", "SW_CustomerOutput", c_softwareCustomerNr );
  }
  else
  {
    /* the read in and the firmly defined sw-number not equal */
    putValueToControl("SetCustomer", "SW_CustomerOutput", "number not equal");
  }
}
```

Tabelle 4.6: Vor- und Nachteile für den Ansatz 4.3.2 – Automatische Auswahl des Kunden über die ECU-Softwarenummer

Vorteile	Dem Entwickler wird manuelles Eingreifen erspart, dadurch fehlerunanfällig
	Sichere Ermittlung des angeschlossenen ECUs über die eindeutige Softwarenummer – Fehlerausgabe bei falscher ECU
Nachteile	Erhöhter Wartungsaufwand, da Softwarenummer manuell im CAPL-Knoten auf aktuellem Stand gehalten werden muss
	Diagnosebeschreibungen für die Plattformhersteller müssen vorhanden sein

Ansatz 4.3.3 – Automatische Auswahl des Kunden über Diagnoseanfrage des passenden Steuergeräts

Für die Ermittlung des angeschlossenen Steuergeräts wird eine Diagnosesitzung an alle in der Plattform vorhanden Steuergeräte über einen „Request“-Befehl angefordert. Listing 4.3 zeigt dabei beispielhaft den Aufbau einer „Default Session“ für einen einzelnen OEM. Da nur das Steuergerät, welches eine physikalische Verbindung zur Restbussimulation hat, einen „Response“ liefern kann, ist ein Rückschluss auf das ECU möglich. Listing 4.4 zeigt dabei die CAPL-Funktion, welche bei einer Antwort für den OEM_1 aufgerufen wird. Innerhalb der Funktion wird die globale Umgebungsvariable gesetzt, die den Kunden festlegt.

Listing 4.3: Anfrage für eine Diagnosesitzung

```
/* Define a diagnostic session variable for the OEM Mahindra              */
/*   -> the diagnostic session variable are declared in the CANdela-file  */
/*      of the OEM Mahindra                                               */
diagRequest PKE_Mahindra.DefaultSession_Start     state_Mahindra_ECU;

/* send a request command to open a diagnostic session with the Mahindra ECU    */
diagSendRequest(state_Mahindra_ECU);
```

Listing 4.4: Response-Funktion für die Diagnosesitzung

```
/*-------------------------------------------------------------------------*/
/* Response of the Mahindra ECU of the default session request              */
/*-------------------------------------------------------------------------*/
on diagResponse PKE_Mahindra.DefaultSession_Start
{
  /* set the the global customer enviroment variable of the OEM Mahindra */
  @customer_env = M327902_MAHINDRA;

  /* set the user instruction in the customer panel */
  setControlPanel();
}
```

Tabelle 4.7: Vor- und Nachteile für den Ansatz 4.3.3 – Automatische Auswahl des Kunden über eine Diagnoseanfrage

Vorteile	Sehr einfache Implementierung und Bedienung
	Kein Wartungsaufwand durch Änderung der Softwarenummer oder Ähnlichem
	Weitere Hersteller können leicht eingepflegt werden
	Dem Entwickler wird manuelles Eingreifen erspart, dadurch fehlerunanfällig
Nachteile	Diagnosebeschreibung für die Plattformhersteller muss vorhanden sein

4.4 Einbinden der Diagnosedienste

Um Zugriff auf die Parameter des Steuergeräts zu bekommen, werden Diagnosebefehle benötigt, die auf dem Diagnoseprotokoll beruhen. Diese Diagnosebefehle sind von den einzelnen Herstellern spezifiziert, sodass ein standardisierter Zugriff erfolgen kann.

Ansatz 4.4.1 – Setzen der Diagnosebeschreibungen beim Simulationsstart

Wird während des Startvorgangs der CANoe-Konfiguration keine Speicherreferenz zur Diagnosebeschreibung gefunden, so kann der Speicherort über ein Fenster manuell ausgewählt und zur Konfiguration hinzugefügt werden. Durch das Abspeichern der Konfiguration wird ebenfalls die Referenz zur Diagnosebeschreibung abgespeichert. Somit muss beim Schließen der CANoe-Konfiguration die Referenz, z. B. über ein Skript, entfernt werden, damit ein manuelles Auswählen beim nächsten Programmstart wieder gewährleistet ist.

Tabelle 4.8: Vor- und Nachteile für den Ansatz 4.4.1 – Setzen der Diagnosebeschreibung beim Simulationsstart

Vorteile	Eindeutige Zuordnung von Diagnosebefehlen zum Kunden-ECU.
	Es werden nur Diagnosebefehle des angeschlossen Steuergeräte-Kunden unterstützt.
Nachteile	Erhöhter Aufwand, da eineDiagnosebeschreibung bei jedem Konfigurationsstart manuell eingefügt werden muss.
	Fehleranfällig durch manuelles Einfügen der Diagnosebeschreibung bei jedem Start der Konfiguration.

Nicht umsetzbar, da CAPL nur sehr begrenzt das bedingte Kompilieren unterstützt und dadurch keine kundenspezifischen Diagnosebefehle innerhalb CAPL verwendet werden können. Es ist nur eine Umsetzung möglich, wenn alle Diagnosebefehle der Plattform-Kunden in einem CANdela-File verwaltet werden.

Ansatz 4.4.2 – Jedem Herstellernetzwerk wird eine eigene Diagnosebeschreibung zugewiesen

Jedes Netzwerk, welches einen Hersteller repräsentiert und für die Kommunikation mit dem spezifischen ECU zuständig ist, bekommt eine separate Diagnosebeschreibung zugewiesen. Abbildung 4.6 zeigt visuell die Zuordnung der Diagnosebeschreibungen für die einzelnen kundenspezifischen Netzwerke. Durch diese direkte Zuweisung zum Kundennetzwerk können die Diagnosebefehle der entsprechenden Diagnosebeschreibung nur in dem inkludierten Netzwerk ausgeführt werden.

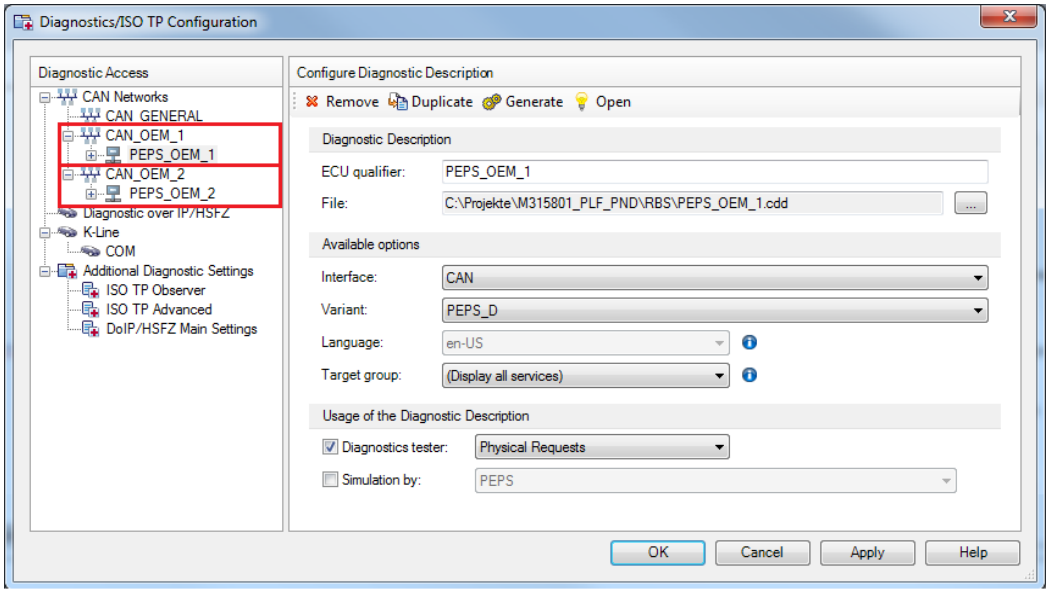

Abbildung 4.6: Zuweisung der kundenspezifischen Diagnosebeschreibung für die einzelnen OEM-Netzwerke

Tabelle 4.9: Vor- und Nachteile für den Ansatz 4.4.2 – Jedem Herstellernetzwerk wird eigene Diagnosebeschreibung zugewiesen

Vorteile	Eindeutige Zuordnung von Diagnosebefehlen zum Kunden-ECU
	Kein Wartungsaufwand durch z. B. Änderungen der Softwarenummer
	Fehlerunanfällig, da jeder Kunden eine eigene Diagnosebeschreibung erhält
	Diagnosebeschreibungen können kundenunabhängig verändert werden
Nachteile	Sorgt für weiteren Overhead in den einzelnen CAPL-Knoten, welche auf die Diagnosebefehle greifen

5 Implementierung

Dieses Kapitel beschreibt die Implementierung der Restbussimulation, welche auf den erstellten Konzepten in Kapitel 4 beruht. Dabei wurden die Vor- und Nachteile der einzelnen Konzepte abgewogen, um eine Simulationsumgebung zu erstellen, welche die funktionalen und nichtfunktionalen Anforderungen bestmöglich erfüllt.

5.1 Aufteilung in separate Herstellernetzwerke

Die einzelnen Plattform-Hersteller werden in separate Netzwerke aufgeteilt. Dies entspricht dem Ansatz 4.1.2. Vorteile, wie jedem Kunden eine eigene spezifische Diagnosebeschreibung zuzuweisen, einfache Deaktivierung kompletter Netzwerke oder bessere Wartbarkeit, sprechen für die Aufteilung in separate Herstellernetzwerke. Durch die klare Struktur der Netzwerke mit der Verbindung zu den jeweiligen Kunden ist eine einfache und schnelle Erweiterung um weitere Plattformhersteller gegeben, was einen deutlichen Vorteil gegenüber einem großen unübersichtlichen Netzwerk, wie in Ansatz 4.1.1 beschrieben, bringt. Die dadurch gewonnene Zeit bei der Implementierung ermöglicht kürzere Entwicklungszyklen, was für das Bestehen am Markt entscheidend ist. Die Vorteile von Ansatz 4.1.1 gegenüber 4.1.2, wie einfachere Einbindung neuer Knoten oder die Verwendung eines einzigen simulierten CAN-Kanals, bringen nur einen signifikant kleineren Mehrwert. Somit sind diese nur bedingt relevant. Da beide erarbeiteten Ansätze den redundanten Code in gemeinsamen Netzwerkknoten bündeln, gibt es in diesem Punkt keinen Vorteil für eines der erarbeiteten Konzepte.

Für Plattformknoten, welche mehrere Hersteller einbinden, findet eine Implementierung in einem allgemeinen Netzwerk statt (siehe Abbildung 4.4). So werden in diesem Netzwerk die gleichen Softwaremodule der verschiedenen Kunden zu einem gemeinsamen Netzwerkknoten zusammengefasst.

Analog dazu findet die Anordnung der Datenbanken statt. Herstellerspezifische Datenbanken, welche nur Informationen eines speziellen Kunden enthalten, werden in dieses Netzwerk eingebunden. Globale Datenbanken, die Informationen mehrerer Hersteller aufweisen, werden dem allgemeinen Netzwerk zugewiesen.

5.1.1 Treibereinstellungen

Um zwischen dem herstellerspezifischen Netzwerk und dem globalen Netzwerk kommunizieren zu können, d. h. den Austausch von CAN-Nachrichten zwischen den beiden simulierten Bussen zu realisieren, müssen diese in der Treibereinstellung dem physikalischen Bus zugewiesen werden. Durch das Hinzufügen von mehreren Netzwerkknoten auf einen physikalischen CAN-Bus wird ein Gateway zwischen den Netzwerken aufgebaut (siehe Abbildung 5.1). Dadurch ist eine interne Kommunikation zwischen den beiden Netzwerken und dem physikalischen Bus möglich. CANoe-intern werden die symbolischen Nachrichten immer auf dem höchstwertigen simulierten Netzwerk angezeigt.

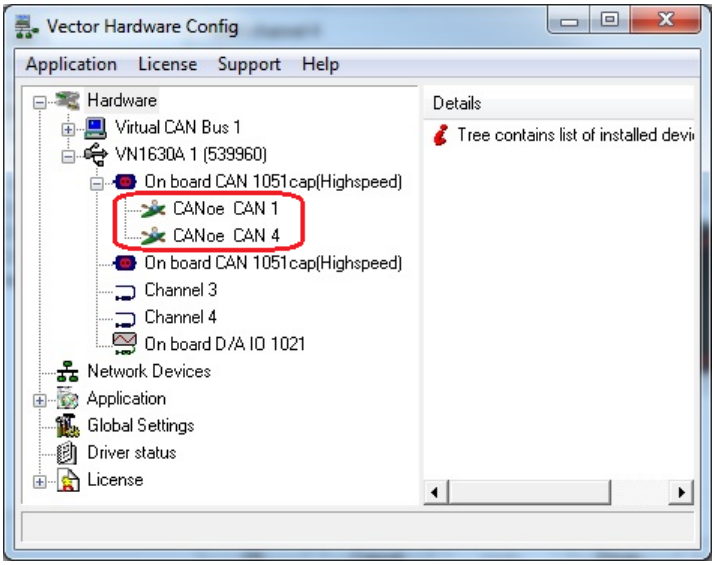

Abbildung 5.1: Die hinzugefügten Kanäle (CAN 1 - globales Netzwerk und CAN 4 - kundenspezifisches Netzwerk) entsprechen je einem Netzwerk innerhalb CANoe.

5.1.2 Qualifizierte Signale und Botschaften

Durch die globale Verwendung der Datenbanken steht jeder Datenbankinhalt jedem Netzwerkknoten zur Verfügung. Dies hat zur Folge, dass es zu redundanten Signalen oder Botschaften kommen kann. Namensräume schaffen hierbei Abhilfe (Listing 5.1). So können Botschaften, Netzwerke sowie zugeordnete Kanäle als Namensräume für Signale verwendet werden, um Konflikte von gleichen Signalnamen zu umgehen. Dadurch wird sichergestellt, dass nur eine Reaktion auf Botschaften oder Signale erfolgt, die in dem entsprechenden Namensraum liegen. Das Zugriffsprinzip mit dem Doppelpunktoperator erfolgt analog zum Namensraumzugriff (Scope-Operator) in C++ [Herol u.a. 2005, S. 38].

Listing 5.1: Namensraumdefinition über sogenannte „Qualifier“

```
/* Access to signals with some differnt qualifier */
$WheelSpeed_BR;                          //Signal
$ESP_Control::WheelSpeed_BR;             //Message + Signal
$CAN_OEM_1::ESP_Control::WheelSpeed_BR;  //Network + Message + Signal
$CAN2::ESP_Control::WheelSpeed_BR;       //Channel + Message + Signal
```

Für die Problemlösung von redundanten Botschaften gibt es drei Möglichkeiten (Listing 5.2, Listing 5.3 und Listing 5.4). Alle Möglichkeiten liefern dasselbe Ergebnis (Abbildung 5.2) und können somit konsistent verwendet werden. Die Kanalzuweisung der Botschaft wird dabei automatisch von CANoe vorgenommen. Dabei wird auf den Speicherort der Datenbank referenziert und daraus ein Rückschluss auf den Kanal des Netzwerks geschlossen. So können über die Namensräume der spezifischen Kunden individuelle Funktionen aufgerufen werden.

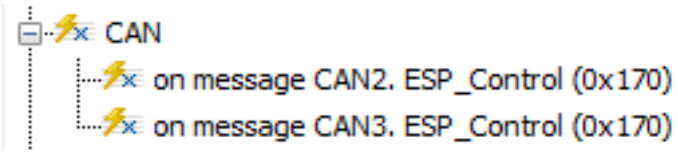

Abbildung 5.2: Kanalzuweisung für redundante Botschaften über Namensraumdefinition

- Möglichkeit 1

Listing 5.2: Verweis auf entsprechende Datenbank

```
/* message event for OEM 1 */
on message OEM_1_Database::ESP_Control
{
  //ESP control message for OEM 1
}

/* message event for OEM 2 */
on message OEM_2_Database::ESP_Control
{
  //ESP control message for OEM 2
}
```

- Möglichkeit 2

Listing 5.3: Verweis auf entsprechenden softwareseitigen Netzwerkkanal

```
/* message event for OEM 1 */
on message CAN2::ESP_Control
{
  //ESP control message for OEM 1
}

/* message event for OEM 2 */
on message CAN3::ESP_Control
{
  //ESP control message for OEM 2
}
```

- Möglichkeit 3

Listing 5.4: Verweis auf das kundenspezifische Netzwerk

```
/* message event for OEM 1 */
on message CAN_OEM_1::ESP_Control
{
  //ESP control message for OEM 1
}

/* message event for OEM 2 */
on message CAN_OEM_2::ESP_Control
{
  //ESP control message for OEM 2
}
```

In dieser Arbeit wurde die Namensraumdeklaration für die kundenspezifischen Signale und Botschaften über das passende Netzwerk implementiert. So ist im Code auf den ersten Blick ersichtlich, zu welchem Kunden die Umgebungsvariable gehört. Da sich Datenbanken und Netzwerkkanäle zur Laufzeit ändern können, wurden diese beiden Möglichkeiten in dieser Arbeit nicht umgesetzt.

Bei der Verwendung der einzelnen Diagnosebeschreibungen für die spezifischen Kunden wird ebenfalls eine Unterscheidung durch die Namensräume getroffen. Das Verfahren hierzu ist analog zu den Namensraumdefinitionen der Botschaften/Signale. Der ECU-Namensraum (ECU qualifier) in der Diagnosebeschreibung sorgt für Eindeutigkeit bei redundanten Diagnosebefehlen.

5.1.3 Deaktivieren von CAPL-Knoten/Netzwerken

Da die kundenspezifischen Netzwerkknoten Senderoutinen besitzen, welche einen Einfluss auf die Buskommunikation haben, müssen diese Knoten deaktiviert werden. Somit ist ihnen die Kommunikation mit dem Netzwerk untersagt. Durch die Unterteilung in Herstellernetzwerke der einzelnen Kunden ist eine Deaktivierung kompletter Netzwerke (siehe Abbildung 4.2) möglich. So bietet hier die Kundenaufteilung neben der Übersichtlichkeit auch eine gute Performance für den Benutzer.

5.2 Dynamische Anpassung der Protokolle CAN und J1939

Um eine dynamische Anpassung der Protokolle gewährleisten zu können, wird der Ansatz 4.3.2 umgesetzt. Der große Vorteil von diesem Ansatz ist es, dass über ein ausführbares Programm die Protokolle angepasst werden ohne dass dabei alle Datenbanken, welche das J1939-Protokoll unterstützen, geöffnet und manuell abgeändert werden müssen. Dadurch wird eine schnellere, effizientere und sicherere Anpassung der Protokolle gewährleistet.

Die Programmausführung erfolgt über einen File-Button im „SetCustomer“-Panel (siehe Abbildung 5.3). Für das Updaten der Übertragungsprotokolle soll das

ausführbare Programm die folgenden Features enthalten, um eine performante Benutzung zu gewährleisten:

- Programm wird über einen dynamischen Pfad innerhalb CANoe aufgerufen.
- Die Datenbanken werden als Textdateien geöffnet und an das zu verwendende Protokoll angepasst.
- Es werden die beiden Protokolle J1939 und CAN durch die Programmdatei unterstützt.

Für die dynamische Anpassung an den Speicherort der Restbussimulation wird auf den Übergabewert der Main-Funktion referenziert. Die einzelnen J1939 Datenbanken werden hingegen statisch im Quellcode definiert (siehe Listing 5.5). Dabei werden sie in Listen verwaltet. Dies bietet den Vorteil, dass man sie sehr einfach erweitern und nacheinander abarbeiten kann [Herol u.a. 2005, S. 491-496]. Nach der Definition aller Datenbanken werden sie nacheinander umbenannt und lesend als Textdatei geöffnet. Nachfolgend werden je nach ausgewähltem Protokoll die Attribute für die Beschreibung der Übertragungsprotokolle angepasst und mit dem originalen Dateinamen abgespeichert. Die umbenannten Pseudo Dateien werden abschließend wieder gelöscht.

Listing 5.5: Einlesen aller Datenbanken, welche sich in einem Ordner befinden

```
/*----------------Include all J1939 dbcs into this list!!-------------------*/
allfiles.push_back((mainstring + "\\DBC\\SXQC\\J1939_sxqc_1.0.dbc"));
allfiles.push_back((mainstring + "\\DBC\\Polaris\\J1939_polaris_1.3.dbc"));

/*--------------------------------------------------------------------------*/
```

5.3 Grafische Auswahl der Hersteller

Für die Auswahl des Kunden wurden alle drei erarbeiteten Ansätze (4.3.1, 4.3.2 und 4.3.3) implementiert und die einzelnen Funktionalitäten in einem Panel

vereint. Die Auswahl des Herstellers ist für die dynamische Anpassung für kundenspezifische Parameter notwendig und wird in Kapitel 4.3 erläutert.

Die manuelle Auswahl (Ansatz 4.3.1) ist entscheidend für die Auswahl des OEM und somit des angeschlossenen ECUs. Punkte wie einfache und schnelle Anbindung weiterer Kunden in die Datenbank und CAPL-Code, intuitive Bedienung und übersichtliche Darstellung des eingestellten Kunden sprechen allesamt für die manuelle Kundenauswahl. Der gesetzte Kunde im Drop-Down-Menü (Abbildung 5.3 – Feld (1)) stellt dabei die globale Zugriffsvariable innerhalb der kompletten Simulation dar. So wird der Aufbau der Diagnosesitzung (Ansatz 4.3.3) lediglich für die Kontrolle zwischen dem manuell eingestellten OEM und dem angeschlossenen ECU verwendet. Im Fehlerfall erfolgt eine Ausgabe an den Benutzer, dass der eingestellte OEM nicht mit dem angeschlossenen ECU übereinstimmt. Das Abfragen der Softwarenummer, wie in Ansatz 4.3.2 beschrieben, ist nur als reine Information für den Entwickler zu verstehen und hat keine Auswirkung auf die Auswahl des OEMs.

Das grafische Panel (Abbildung 5.3) enthält alle wichtigen Funktionen, welche für die Parametereinstellungen der spezifischen Kunden notwendig sind. Die Logik des Panels wurde im CAPL-Knoten SetCustomer.can implementiert.

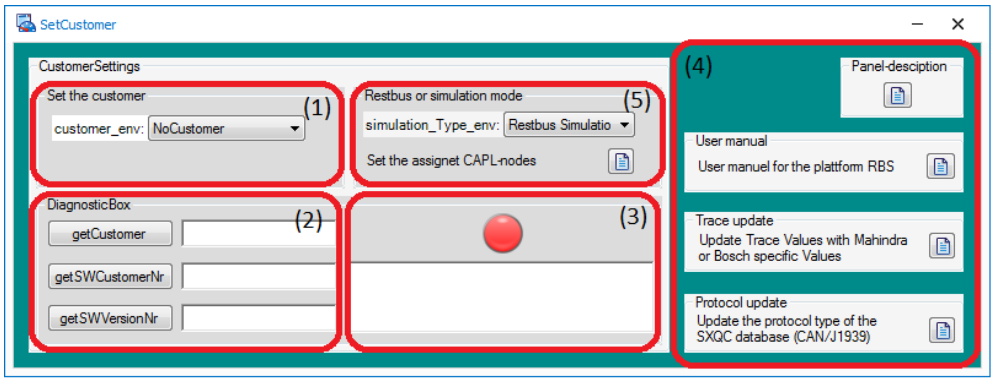

Abbildung 5.3: Grafisches Panel für die Auswahl der spezifischen Kundenparameter sowie das angeschlossene ECU.

(1) Manuelles Einstellen des Kunden

In der Datenbank *customer.dbc* befindet sich die Umgebungsvariable *customer_env*, welche für die Verwaltung der Plattformkunden zuständig ist. Eine Wertetabelle sorgt für eine klare Beschreibung durch die Projektnummer der einzelnen Kunden. Der Wert dieser Umgebungsvariablen ist der Referenzpunkt in der kompletten Simulation, weshalb sich das angeschlossene ECU mit dem Kunden decken muss. Dabei wird explizit auf eine Umgebungsvariable einer Datenbank verwiesen. Dadurch ist die Möglichkeit gegeben, die Kundenvariable als Referenz für Skripte, auch außerhalb von CANoe, zu verwenden. Systemvariablen hingegen lassen nur einen Zugriff innerhalb von CANoe zu. Listing 5.6 zeigt exemplarisch die Abfrage der Umgebungsvariable in Bezug auf den eingestellten Kunden. Dabei wird der Identifier einer Botschaft an den eingestellten Kunden und somit an das angeschlossene Steuergerät dynamisch angepasst.

Listing 5.6: Abfrage der Umgebungsvariable in Bezug auf den eingestellten Kunden

```
/* check the customer and set the ID identifier */
if(@customer_env == BOSCH)
{
  response_msg.id  = 0x611;     //set the response ID for BOSCH
}
else if(@customer_env == MAHINDRA)
{
  response_msg.id  = 0x602;    //set the response ID for Mahindra
}
else
{
  retValue = -1;
}
```

(2) Automatische und manuelle Abfrage des angeschlossenen ECU sowie Auslesen der Softwarenummer

Beim Starten der Simulation werden automatisch, nach einem Delay von 300 ms, verschiedene Diagnoseanfragen über die Diagnoseschnittstelle an das angeschlossene ECU gesendet. Dabei werden die drei Anfragen

- Diagnosesitzung (für das Setzen des Kunden)
- Software-Kundennummer
- Software-Versionsnummer

an das angeschlossene Steuergerät gesendet. Die Abfragen können zur Laufzeit der Simulation manuell durch die einzelnen Buttons durchgeführt werden (siehe Listing 5.7). Dabei wird für die Ermittlung des Herstellers eine Diagnosesitzung an alle Hersteller gesendet, dies entspricht dem Vorgehen des Ansatzes 4.3.3. Da nur das angeschlossene ECU und somit nur ein Kunde antworten kann, kann dieser über die globale Umgebungsvariable gesetzt werden. Für den Aufbau einer Diagnosesitzung ist die Deklaration in CANoe für die entsprechende Kundendiagnosebeschreibung vorzunehmen.

Listing 5.7: Manuelles Senden von Diagnoseanfragen

```
/*------------------------------------------------------------------------*/
/* Event function of the system variable, which is linked to              */
/* button "getCustomer" in the "SetCustomer-Panel                         */
/*------------------------------------------------------------------------*/
on sysvar Customer_SysVar::CheckCustomer
{
  /* send diagnostic request to all platform customer ECUs */
  diagSendRequest(state_Mahindra_ECU);
  diagSendRequest(state_BOSCH_ECU);
  //...more platform customers
}
```

(3) Nachrichtenfenster mit hilfreichen Hinweisen für den Entwickler

Dieses Fenster gibt dem Entwickler Information darüber, welche Knoten er in den einzelnen Netzwerken deaktivieren muss sowie über eventuell auftretende Fehler beim Simulationsstart. Stimmt das angeschlossene physikalische

Steuergerät mit dem softwareseitig eingestellten Kunden nicht überein, so erfolgt ein Hinweis über das Ausgabefenster an den Benutzter. Zusätzlich werden den Benutzern durch eine rot blinkende Signallampe Fehlerzustände angezeigt.

(4) Verschiedene Links für Skripte und für kundenspezifische Einstellungen

Der Paneleditor in CANoe bietet die Möglichkeit, externe Dateien über eine spezielle Schaltfläche aufzurufen. Der Speicherort der aufzurufenden Datei ändert sich dabei relativ zum Speicherort der Konfiguration ab. Dadurch kann man den kompletten Dateiordner verschieben, ohne dabei die Referenz zur entsprechenden Datei zu verlieren.

- Beschreibung der Funktionen und Benutzung des Panels.
- Allgemeine Bedienungsanleitung für die Benutzung und Konfiguration der Plattform-Restbussimulation.
- Anpassen von Wertetabellen in den Datenbanken für spezifische Kunden (diese Funktion ist nicht Teil dieser Arbeit[1] und wird deshalb nicht näher beschrieben).
- Automatisches Updaten der Übertragungsprotokolle (CAN – J1939). Die Funktionalität dieses Skripts ist in Kapitel 5.2 beschrieben.

(5) Setzen der Netzwerke über ein externes Skript

Um die kundenspezifischen Anpassungen bei einem Kundenwechsel[2] zu vereinfachen, werden diese von einem Skript übernommen. Dieser automatisierte Prozess beinhaltet das Aktivieren/Deaktivieren, das Setzen der

[1] Das Anpassen der Wertetabellen über ein Skript wird im Rahmen eines Praxissemesters parallel zu dieser Arbeit durchgeführt.
[2] Ersetzen von dem realen ECU durch ein ECU eines anderen Herstellers

Diagnosefenster sowie die Treibereinstellungen (Kapitel 5.1.1). Dabei wird auf den eingestellten Kunden im Drop-Down-Menü referenziert und dementsprechend werden die kundenspezifischen Anpassungen vorgenommen. Hier bietet die Umgebungsvariable den Vorteil, dass auch außerhalb von CANoe auf sie zugegriffen werden kann. Zudem wird die Performance bei einem Kundenwechsel deutlich gesteigert und die Fehlerrate bei der Bedienung gesenkt. Erweiterungen oder Änderungen des Netzwerkaufbaus haben Anpassungen des Skripts zur Folge, sodass externe Dateien abgeändert werden müssen. Die Umsetzung und Implementierung dieses Skripts war nicht Bestandteil dieser Arbeit.

6 Bewertung und Inbetriebnahme der Restbussimulation

Anhand der gestellten Anforderungen sowie Grundfunktionen wird eine Bewertung der Plattform-Restbussimulation durchgeführt. Für die Inbetriebnahme der Restbussimulation kommt eine Integrationsbox zum Einsatz, welche die Schnittstelle zwischen dem CAN-Bus und dem realen Steuergerät darstellt. Zudem werden durch die Integrationsbox fahrzeugrelevante Aktoren und Sensoren wie

- Türgriffsensoren für das Ver- und Entriegeln,
- Start-Stopp-Button,
- Visuelle Anzeige der verschiedenen Fahrzeugklemmen sowie
- Anschluss der Fahrzeugantennen

eingebunden, um einen Funktionstest der Steuergerätesoftware sowie der Restbussimulation zu ermöglichen.

Für das Ausrollen[1] der Plattform-Restbussimulation müssen alle Testanforderungen und Grundfunktionen in den Tabelle 6.1 und Tabelle 6.2 erfüllt sein, sodass die Grundfunktionalität der Simulation gewährleistet ist. Ansonsten müssen Nachbesserungen durchgeführt werden. Modulspezifische Tests werden durch die zuständigen Modulsoftwareentwickler durchgeführt und werden nicht für die Verifizierung der Plattform-Konfiguration benötigt.

[1] Die Restbussimulation wird für alle Plattform-Modulentwickler öffentlich zugänglich gemacht.

6.1 Bewertung der funktionalen Anforderungen

Für die Inbetriebnahme der Plattform-Restbussimulation müssen die nachfolgenden Anforderungen erfüllt sein. Tabelle 6.1 zeigt alle getesteten Anforderungen bezüglich der gestellten Evaluationskriterien für zwei Plattformkunden.

1. **Herstellerauswahl über das grafische Panel**
   - **1.1 Manuelle Kundenauswahl**

     Kunde soll manuell über das Drop-Down-Menü ausgewählt werden. Der in dem Menü eingestellte Kunde stellt den globalen Zugriffpunkt in der kompletten Konfiguration dar.
   - **1.2 Automatische Kundenauswahl**

     Der Kunde soll automatisch über den Aufbau einer Diagnosesitzung zum angeschlossenen ECU ermittelt werden. Folglich soll der Kunde über ein Ausgabefenster angezeigt und entsprechend im Drop-Down-Menü gesetzt werden. Der in dem Menü eingestellte Kunde stellt wiederum den globalen Zugriffpunkt in der kompletten Konfiguration dar.

2. **Verwenden der Diagnosebeschreibung**

   Um wichtige Fahrzeugparameter zu setzen und abzufragen wird die Diagnosebeschreibung der einzelnen Hersteller benötigt. Dabei sollen verschiedene Diagnosesitzungen aufgebaut, verschiedene Sicherheitsschlüssel korrekt berechnet sowie Diagnosebefehle korrekt abgesendet und empfangen werden.

3. **Protokollupdate für CAN- und J1939-Netzwerke**

   Ein externes Skript, welches über ein CANoe-Panel ausgeführt werden kann, soll die entsprechenden Übertragungsprotokolle innerhalb der Datenbanken abändern. Erfolgreich angepasste Datenbanken sollen im Konsolenfenster angezeigt werden.

4. Testen weiterer wichtiger Funktionen[2]

4.1 Schlüsselanlernen für die einzelnen Plattformkunden

Hierfür sollen spezielle Ablaufsequenzen ausgeführt werden. Sie werden über Makros oder CAPL-Programme implementiert.

4.2 Ent- und Verriegelung des Fahrzeugs

Das Ent- und Verriegeln soll über die Taster der Integrationsbox erfolgen. Die Bewertung erfolgt über die Zustandsänderung von den entsprechenden Signalwerten innerhalb CANoe und kann durch das „Graphic-Window“ visualisiert werden.

4.3 Grundlegende Keyless-Funktionen

Kommunikationstests zwischen Schlüssel und Steuergerät sollen durch LF- und HF-Kommunikation sowie das manuelle Ent- und Verriegeln verifiziert werden. Die Bewertung soll über die entsprechende Signaländerung erfolgen, welche in CANoe nachvollzogen wird.

4.4 Start-/Stopp-Funktionen des Fahrzeugs

Die Evaluation soll über die Signaländerung sowie Klemmenzustandsänderung durchgeführt werden.

[2] Die Implementierung dieser Funktionen war nicht Bestandteil dieser Arbeit. Sie wurden jedoch für die Verifizierung der Plattformkonfiguration hinzugenommen.

Tabelle 6.1: Bewertung der funktionalen Anforderungen der Restbussimulation für zwei Kunden.

	Bewertungskriterien	BOSCH	Mahindra
1.1	Manuelle Kundeauswahl	✓	✓
1.2	Automatische Kundenauswahl	✓	✓
2	Verwendung der Diagnosebeschreibung	✓	✓
3	Protokollupdate für CAN und J1939 Netzwerke	✓	✓
4.1	Schlüsselanlernen für einzelne Plattformkunden	✓	✓
4.2	Ent- und Verriegelung des Fahrzeugs	✓	✓
4.3	Grundlegende Keyless-Funktionen	✓	✓
4.4	Start-/Stopp-Funktion des Fahrzeugs	✓	✓

6.2 Bewertung der nichtfunktionalen Anforderungen

Die nichtfunktionalen Anforderungen, die wichtige Qualitätsmerkmale der Konfiguration enthalten, werden durch die nachfolgenden Punkte evaluiert. Die Evaluationsergebnisse der Testpunkte werden in Tabelle 6.2 dargestellt.

1. **Übersichtlichkeit der CANoe-Konfiguration**
   - **1.1 Aufteilung in kundenabhängige Netzwerke**

     Jeder Kunde soll einem kundenabhängigen Netzwerk zugeteilt sein. Dabei soll auf eine einheitliche Struktur der Netzwerke geachtet werden.

1.2 Zuordnung der Namensräume

Für die korrekte Zuordnung der Botschaften, Signale und Diagnosebefehle zu ihren herstellerspezifischen Netzwerken soll eine manuelle Kontrolle erfolgen. Abbildung 5.2 stellt dabei die Zuordnung zwischen definiertem Namensraum der Botschaft und einem CAN-Netzwerk, welches über eine Kanalnummer identifiziert wird, dar. Zudem sollen keine Warnungen beim Kompilieren von Netzwerkknoten auftreten, welche auf redundant angelegte Botschaften, Signale oder Diagnosebefehle verweisen.

2. Erweiterbarkeit der Plattform-Konfiguration

Die Plattform-Konfiguration soll um einen weiteren Kunden ergänzt werden. Dabei sollen alle notwendigen Module eingebunden und nach ihrer Funktion getestet werden. Die Einbindung der Module soll nach der Struktur der bisherigen Kunden erfolgen. Die Implementierung dafür ist in Kapitel 5 aufgeführt.

3. Echtzeitfähigkeit der Restbussimulation

Die Restbussimulation soll die Echtzeit der CAN-Botschaften einhalten. Dabei soll es zu keinem Überlauf der CAN-Frame Buffer des CAN-Controllers kommen, was einen Error Frame zur Folge hätte.

Tabelle 6.2: Bewertung der nichtfunktionalen Anforderungen der RBS

	Bewertungskriterien	Test erfolgreich
1.1	Aufteilung in kundenabhängige Netzwerke	✓
1.2	Zuordnung der Namensräume	✓
2	Erweiterbarkeit der Plattform-Konfiguration	✓
3	Echtzeitfähigkeit der Restbussimulation	✓

7 Fazit und Ausblick

Ziel dieser Abschlussarbeit war es, eine Plattform-Restbussimulation für den Automotive-Bereich mit dem Software-Tool CANoe zu erstellen. Dabei sollten verschiedene OEMs durch eine einzige Simulationsumgebung abgedeckt werden. Da jeder OEM spezifische Netzwerkknoten besitzt, müssen diese dynamisch an das physikalisch angeschlossene Fahrzeugsteuergerät angepasst werden. So wurden zu Beginn der Arbeit Grundlagenrecherchen in CANoe, Aufbau und Funktionsweise einer Restbussimulation sowie dem Keyless-Prinzip betrieben. Des Weiteren wurde auf die gängigen Bussysteme und Übertragungsprotokolle in einem Fahrzeug, insbesondere das CAN-Protokoll, eingegangen. Bei der Implementierungsphase wurde auf die Vor- und Nachteile der zuvor erstellten Konzepte eingegangen und diese wurden entsprechend bewertet. Dabei wurde für jeden OEM, welcher durch die Plattform-Restbussimulation unterstützt wird, ein eigenes Netzwerk angelegt. Dieses beinhaltet die spezifischen Netzwerkknoten der einzelnen Plattform-OEMs, welche die relevanten ECUs des Fahrzeugs simulieren sowie die kundenspezifischen Datenbanken. Kunden übergreifende Netzwerkknoten und Datenbanken werden in einem globalen Netzwerk verwaltet. Durch diese klare Struktur kann die Plattform-Restbussimulation leicht um weitere zukünftige Kunden ergänzt werden. Zusätzlich wurden die erstellten Konzepte für die Anpassung der Übertragungsprotokolle CAN und J1939 sowie die dynamische Kundenauswahl erfolgreich implementiert. Um eine Aussage über das Erreichen der Ziele treffen zu können, wurde die Simulation in Betrieb genommen und anhand der erstellten Anforderungen bewertet. Die dabei eingesetzte Integrationsbox stellt dem Entwickler die fahrzeugrelevanten Sensoren und Aktoren, welche für die Funktionsfähigkeit des schlüssellosen Komfortzugangs notwendig sind, zur Verfügung. Durch die erfolgreiche Bewertung und

Inbetriebnahme der Restbussimulation konnten alle vorgegebenen Ziele verifiziert werden. Nach der erfolgreichen Verifizierung wurden die kundenspezifischen Restbussimulationen durch die globale Restbussimulation ersetzt. Diese wird bereits von den Plattform-Entwicklern für die Fahrzeugsimulation der Steuergeräte verwendet.

Aufbauend auf dieser Abschlussarbeit kann eine Reihe von Erweiterungen für die Plattform-Restbussimulation vorgenommen werden. Nachfolgend werden zwei, meiner Meinung nach wichtige, Punkte beschrieben, welche in naher Zukunft umgesetzt werden sollten.

Mit den grafischen Benutzeroberflächen wird dem Entwickler ein mächtiges, übersichtliches und zugleich einfach zu bedienendes Werkzeug geboten, welches ihm viele Möglichkeiten eröffnet; vorausgesetzt, die grafischen Benutzeroberflächen sind innerhalb der kompletten Simulationsumgebung klar strukturiert aufgebaut, sodass eine intuitive und einheitliche Benutzung durch die verschiedenen Entwickler gewährleistet ist. Da dies in der aktuellen RBS nur zum Teil der Fall ist, sollten diese in naher Zukunft auf einen einheitlichen Stand gebracht werden. Dabei sollten redundante Panels der unterschiedlichen OEMs in Plattformpanels zusammengefasst werden.

Des Weiteren sollten in Zukunft Netzwerkknoten innerhalb CANoe möglichst modular aufgebaut werden. So ist die Möglichkeit gegeben, die einzelnen Softwaremodule leicht und schnell in andere Netzwerke zu übernehmen. Dadurch kann die Implementierungszeit der Restbussimulation weiter reduziert werden. Um die Übersichtlichkeit und Wartbarkeit der Restbussimulation weiter zu steigern, ist es zudem sinnvoll, den CAPL-Code, welcher für die einzelnen Netzwerkknoten die Logik enthält, nach fest definierten Code-Richtlinien aufzubauen, um den Code einheitlicher zu formatieren.

Mit diesen Ansätzen kann die Effizienz bei künftigen Erweiterungen und Anpassungen an weitere Kunden noch weiter gesteigert werden.

Literaturverzeichnis

[Albrecht und Decker 2012]. (März 2012). *Schnelle Wege zur Restbussimulation.* Abgerufen am 04. Oktober 2019 von Vector GmbH: https://www.all-electronics.de/schnelle-wege-zur-restbussimulation/

[Borgeest 2014]. (2014). *Elektronik in der Fahrzeugtechnik* (3 Ausg.). (K. Borgeest, Hrsg.) Wiesbaden, ISBN 978-3-8348-1642-9: Springer Vieweg.

[CANBUS 2018]. (9. Oktober 2018). *Lernmodul - Einführung in CAN*. Abgerufen am 15. November 2019 von Vector Informatik GmbH: https://elearning.vector.com/mod/page/view.php?id=111

[CANdela 2019]. (2019). *CANdela Studio.* Abgerufen am 6. Dezember 2019 von Vector Informatik GmbH: https://assets.vector.com/cms/content/products/candelastudio/docs/CANdelaStudio_FactSheet_DE.pdf

[CANoe 2018]. (04. November 2019). *CANoe Produktinformation.* Abgerufen am 04. Dez. 2019 von Vector Informatik GmbH: https://assets.vector.com/cms/content/products/canoe/canoe/docs/Product%20Informations/CANoe_ProductInformation_DE.pdf

[Ernst u.a. 2015]. (2015). *Grundkurs Informatik* (5 Ausg.). (H. Ernst , J. Schmidt, & G. Beneken, Hrsg.) Wiesbaden, ISBN 978-3-658-01627-2: Springer Vieweg.

[HELLA 2019]. (2019). *Keyless Go*. Abgerufen am 12. November 2019 von Hella.de: https://www.hella.com/techworld/de/Technik/Elektrik-Elektronik/Keyless-Go-3195/

[Herol u.a. 2005]. (2015). *C++, UML und Desgin Pattern.* (H. Herol, M. Klar, & S. Klar, Hrsg.) München, ISBN: 3-8273-2267-7: Addison-Wesly Verlag.

[ISO 14229 UDS]. (01. Dezember 2006). *Road vehicles - Unified diagnostic services (UDS)*. Abgerufen am 09. Januar 2020 von ISO 14229: http://read.pudn.com/downloads191/doc/899044/ISO+14229+(2006).pdf

[ISO/DIS 14230 KWP2000]. (16. Dezember 1996). *Road Vehicles - Diagnostic Systems (KWP2000).* Abgerufen am 09. Januar 2020 von ISO/DIS 14230: http://www.internetsomething.com/kwp/KWP2000%20ISO%2014230-2%20KLine%20.pdf

[J1939 2018]. (23. November 2018). *Lernmodul J1939*. Abgerufen am 15. November 2019 von Vector Informatik GmbH: https://elearning.vector.com/mod/page/view.php?id=253

[KUNBUS 2019]. (2019). *Grundlagen digitaler Bussysteme und wesentliche Grundbegriffe*. Abgerufen am 5. Dezember 2019 von KUNBUS: https://www.kunbus.de/grundlagen-digitaler-bussysteme-und-wesentliche-grundbegriffe.html

[Lobmeyer und Marktl 2014 a]. (Februar 2014). *Steuergerätetests effizienter programmieren - Teil 1: CAPL Basics.* Abgerufen am 3. August 2019 von Vector Informatik GmbH: https://assets.vector.com/cms/content/know-how/_technical-articles/CAPL_1_CANNewsletter_201406_PressArticle_DE.pdf

[Lobmeyer und Marktl 2014 b]. (März 2014). *Steuergerätetests effizienter programmieren - Teil 2: CAPL besser verstehen und effektiv anwenden*. Abgerufen am 5. Dezember 2019 von Vector Infomatik GmbH: https://assets.vector.com/cms/content/know-how/_technical-articles/CAPL_2_CANNewsletter_201409_PressArticle_DE.pdf

[Lobmeyer und Marktl 2014 c]. (April 2014). *Steuergerätetests effizienter programmieren - Teil 3: CAPL für Fortgeschrittene*. Abgerufen am 5. Dezember 2019 von Vector Informatik GmbH: https://assets.vector.com/cms/content/know-how/_technical-articles/CAPL_3_CANNewsletter_201411_PressArticle_DE.pdf

[OEM 1985]. (1985). *Defintion eines OEM*. Abgerufen am 6. Dezember 2019 von Wirtschaftslexikon24.com: http://www.wirtschaftslexikon24.com/d/oem/oem.htm

[SCHRADER 2010]. (15. Juli 2010). *Smartphone als Car-Computer und Diagnosewerkzeug*. Abgerufen am 5. Dezember 2019 von CNET: https://www.cnet.de/41534874/smartphone-als-car-computer-und-diagnosewerkzeug-obd-auf-dem-handy/

[Sutherland und Schwaber 2013]. (November 2017). *The Scrum Guide.* Abgerufen am 8. Januar 2020 von Scrum Guide: https://www.scrumguides.org/scrum-guide.html

[Vector Informatik 2019]. (09. Oktober 2018). *Einführung in CAN*. Abgerufen am 04. Dezember 2019 von https://elearning.vector.com/mod/page/view.php?id=111

[Zimmermann und Schmittgall 2014]. (2014). *Bussysteme in der Fahrzeugtechnik* (5 Ausg.). (W. Zimmermann, & R. Schmidgall, Hrsg.) Wiesbaden, ISBN 978-3-658-02418-5: Springer Vieweg.

Eidesstattliche Erklärung

Hiermit versichere ich eidesstattlich, dass die vorliegende Arbeit mit dem Thema

Handling einer Plattform-Restbussimulation im Automotive-Bereich mit einer dynamischen Anpassung an spezifische Kunden

von mir selbstständig und ohne unerlaubte fremde Hilfe angefertigt worden ist, insbesondere, dass ich alle Stellen, die wörtlich oder annähernd wörtlich oder dem Gedanken nach aus Veröffentlichungen, unveröffentlichten Unterlagen und Gesprächen entnommen worden sind, als solche an den entsprechenden Stellen innerhalb der Arbeit durch Zitate kenntlich gemacht habe, wobei in den Zitaten jeweils der Umfang der entnommenen Originalzitate kenntlich gemacht wurde. Die Arbeit lag in gleicher oder ähnlicher Fassung noch keiner Prüfungsbehörde vor und wurde bisher nicht veröffentlicht.

Rietheim-Weilheim, 20.02.2020

_____

Vorname Nachname